PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 9/44, 17/30

A1

(11) International Publication Number: WO 98/06033

(43) International Publication Date: 12 February 1998 (12.02.98)

(21) International Application Number: PCT/US97/13817

(22) International Filing Date: 8 August 1997 (08.08.97)

(30) Priority Data:
60/023,373 8 August 1996 (08.08.96) US

(71) Applicant: AGRANAT SYSTEMS, INC. [US/US]; 1345 Main Street, Waltham, MA 02154 (US).

(72) Inventors: AGRANAT, Ian, D.; 108 Jericho Road, Weston, MA 02193 (US). GIUSTI, Kenneth, A.; 16 Josiah Drive, Upton, MA 01568 (US). LAWRENCE, Scott, D.; 560 Old Marlboro Road, Concord, MA 01742-4042 (US).

(74) Agent: ENGELSON, Gary, S.; Wolf, Greenfield & Sacks, P.C., 600 Atlantic Avenue, Boston, MA 02210 (US).

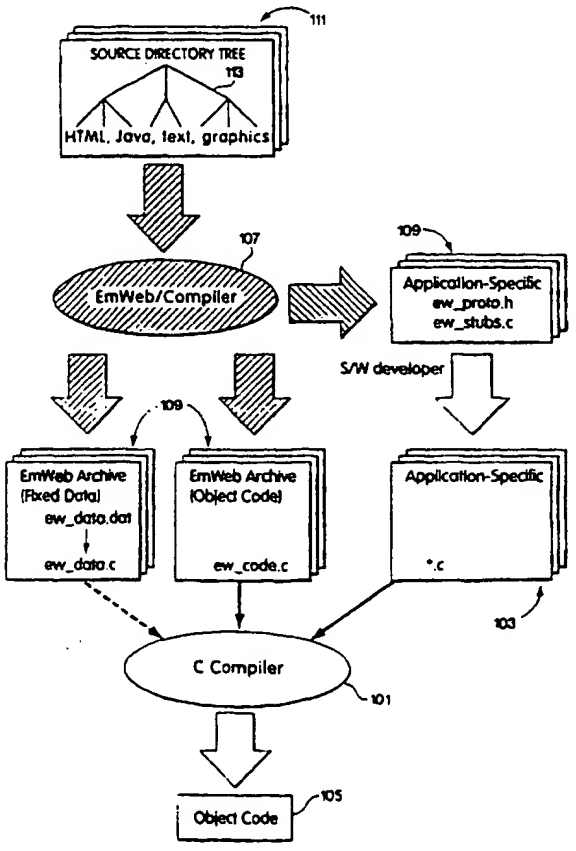
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: EMBEDDED WEB SERVER

(57) Abstract

An embedded graphical user interface employs a World-Wide-Web communications and display paradigm. The development environment includes an HTML compiler which recognizes and processes a number of unique extensions to HTML. The HTML compiler produces an output which is in the source code language of an application to which the graphical user interface applies. A corresponding run-time environment includes a server which serves the compiled HTML documents to a browser.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

EMBEDDED WEB SERVER

COPYRIGHT NOTICE

TABLES

Table A is a functional specification of the Agranat Systems, Inc. EmWeb™ product; and

Table B is a source code listing in the C programming language of header files defining data structures representing an archive of content as used by the EmWeb™ product.

The noted tables are provided to show an example of a particular embodiment of the invention incorporating features thereof which are described in detail below.

CROSS-REFERENCE TO RELATED APPLICATION

Priority is claimed under 35 U.S.C. §119(e) to the inventors' Provisional U.S. Patent Application Serial No. 08/023,373, entitled EXTENDED LANGUAGE COMPILER AND RUN TIME SERVER, filed August 8, 1996, now pending. The inventors' above-identified provisional U.S. patent application is incorporated herein by reference.

1. Field of the Invention

The present invention relates generally to graphical user interfaces (GUIs), i.e. user interfaces in which information can be presented in both textual form and graphical form. More particularly, the invention relates to GUIs used to control, manage, configure, monitor and diagnose software and hardware applications, devices and equipment using a World-Wide-Web client/server communications model. Yet more particularly, the invention relates to methods and apparatus for developing and using such GUIs based on a World-Wide-Web client/server communications model.

2. Related Art

Many modern communications, entertainment and other electronic devices require or could benefit from improved local or remote control, management, configuration, monitoring and diagnosing. It is common for such devices to be controlled by a software application program specifically written for each device. The design of such a device includes any hardware and operating environment software needed to support the application, which is then referred to as an embedded application, because it is embedded within the device. Embedded application programs are generally written in a high-level programming language such as C, C++, etc., referred to herein as a native application programming language. Other languages suitable to particular uses may also be employed. The application program communicates with users through a user interface, generally written in the same high-level language as the application.

The representation of an application in a native application programming language is referred to as the application program source code. A corresponding representation which can be executed on a processor is referred to as an executable image.

Before an application written in a high-level language can be executed it must be compiled and linked to transform the application source code into an executable image. A compiler receives as an input a file containing the application source code and produces as an output a file in a format referred to as object code. Finally, one or more object code files are linked to form the executable image. Linking resolves references an object module may make outside of that object module, such as addresses, symbols or functions defined elsewhere.

Source code may also define arrangements by which data can be stored in memory and conveniently referred to symbolically. Such defined arrangements are referred to as data structures because they represent the physical arrangement of data within memory, i.e., the structure into which the data is organized.

Most commonly, remote control, management, configuration, monitoring and diagnosing applications employ unique proprietary user interfaces integrated with the application software and embedded into the device. Frequently these user interfaces present and receive information in text form only. Moreover, they are not portable, generally being designed to operate on a specific platform, i.e., combination of hardware and software. The devices for which control, management, configuration and diagnosing are desired have only limited run-time resources available, such as memory and long-term storage space. Proprietary interfaces are frequently designed with such limitations to data presentation, data acquisition and portability because of

the development costs incurred in providing such features and in order to keep the size and run-time resource requirements of the user interface to a minimum. Since each user interface tends to be unique to the particular remote control, management, configuration, monitoring or diagnosing function desired, as well as unique to the operating system, application and hardware platform upon which these operations are performed, significant time and/or other resources may be expended in development. Graphics handling and portability have therefore been considered luxuries too expensive for most applications.

However, as the range of products available requiring control, management, configuration, monitoring or diagnosing increase, such former luxuries as graphical presentation and portability of the interface from platform to platform have migrated from the category of luxuries to that of necessities. It is well known that information presented graphically is more quickly and easily assimilated than the same information presented as text. It is also well known that a consistent user interface presented by a variety of platforms is more likely to be understood and properly used than unique proprietary user interfaces presented by each individual platform. Therefore, portable GUIs with low run-time resource requirements are highly desirable.

With the growing popularity and expansion of the Internet, one extremely popular public network for communications between computer systems, and development of the World-Wide-Web communication and presentation model, a new paradigm for communication of information has emerged.

The World-Wide-Web and similar private architectures such as internal corporate LANs. provide a "web" of interconnected document objects. On the World-Wide-Web, these document objects are located on various sites on the global Internet. The World-Wide-Web is also described in "The World-Wide Web, " by T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, Communications of the ACM, 37 (8), pp. 76-82, August 1994, and in "World Wide Web: The Information Universe," by Berners-Lee, T., et al., in Electronic Networking: Research, Applications and Policy, Vol. 1, No. 2, Meckler, Westport, Conn., Spring 1992. On the Internet, the World-Wide-Web is a collection of documents (i.e., content), client software (i.e., browsers) and server software (i.e., servers) which cooperate to present and receive information from users. The World-Wide-Web is also used to connect users through the content to a variety of databases and services from which information may be obtained. However, except as explained below, the World-Wide-Web is based principally on static information contained in the content documents available to the browsers through the servers. Such a limitation would

make the World-Wide-Web paradigm useless as a GUI which must present dynamic information generated by a device or application.

The World-Wide-Web communications paradigm is based on a conventional client-server model. Content is held in documents accessible to servers. Clients can request, through an interconnect system, documents which are then served to the clients through the interconnect system. The client software is responsible for interpreting the contents of the document served, if necessary.

Among the types of document objects in a "web" are documents and scripts. Documents in the World-Wide-Web may contain text, images, video, sound or other information sought to be presented, in undetermined formats known to browsers or extensions used with browsers. The presentation obtained or other actions performed when a browser requests a document from a server is usually determined by text contained in a document which is written in Hypertext Mark-up Language (HTML). HTML is described in HyperText Markup Language Specification - 2.0, by T. Berners-Lee and D. Connolly, RFC 1866, proposed standard, November 1995, and in "World Wide Web & HTML," by Douglas C. McArthur, in Dr. Dobbs Journal, December 1994, pp. 18-20, 22, 24, 26 and 86. HTML documents stored as such are generally static, that is, the contents do not change over time except when the document is manually modified. Scripts are programs that can generate HTML documents when executed.

HTML is one of a family of computer languages referred to as mark-up languages. Mark-up languages are computer languages which describe how to display, print, etc. a text document in a device-independent way. The description takes the form of textual tags indicating a format to be applied or other action to be taken relative to document text. The tags are usually unique character strings having defined meanings in the mark-up language. Tags are described in greater detail, below.

HTML is used in the World-Wide-Web because it is designed for writing hypertext documents. The formal definition is that HTML documents are Standard Generalized Markup Language (SGML) documents that conform to a particular Document Type Definition (DTD). An HTML document includes a hierarchical set of markup elements, where most elements have a start tag, followed by content, followed by an end tag. The content is a combination of text and nested markup elements. Tags are enclosed in angle brackets ('<' and '>') and indicate how the document is structured and how to display the document, as well as destinations and labels for hypertext links. There are tags for markup elements such as titles, headers, text attributes such as

bold and italic, lists, paragraph boundaries, links to other documents or other parts of the same document, in-line graphic images, and many other features.

For example, here are several lines of HTML:

```
Some words are <B>bold</B>, others are <I>italic</I>. Here we start a new
paragraph.<P>Here's a link to
the <A HREF="http://www.agranat.com">Agranat Systems, Inc.</A> home
page.
```

This sample document is a hypertext document because it contains a "link" to another document, as provided by the "HREF=." The format of this link will be described below. A hypertext document may also have a link to other parts of the same document. Linked documents may generally be located anywhere on the Internet. When a user is viewing the document using a client program called a Web browser (described below), the links are displayed as highlighted words or phrases. For example, using a Web browser, the sample document above would be displayed on the user's screen as follows:

> Some words are **bold**, others are *italic*. Here we start a new paragraph.
> Here's a link to Agranat Systems, Inc. home page.

In the Web browser, the link may be selected, for example by clicking on the highlighted area with a mouse. Selecting a link will cause the associated document to be displayed. Thus, clicking on the highlighted text "Agranat Systems, Inc." would display that home page.

Although a browser can be used to directly request images, video, sound, etc. from a server, more usually an HTML document which controls the presentation of information served to the browser by the server is requested. However, except as noted below, the contents of an HTML file are static, i.e., the browser can only present a passive snapshot of the contents at the time the document is served. In order to present dynamic information, i.e., generated by an application or device, or obtain from the user data which has been inserted into an HTML-generated form, conventional World-Wide-Web servers use a "raw" interface, such as the common gateway interface (CGI), explained below. HTML provides no mechanism for presenting dynamic information generated by an application or device, except through a raw interface, such as the CGI. Regarding obtaining data from the user for use by the application or device, although standard HTML provides a set of tags which implement a convenient mechanism for serving interactive forms to the browser, complete with text fields, check boxes

and pull-down menus, the CGI must be used to process submitted forms. Form processing is important to remote control, management, configuration, monitoring and diagnosing applications because forms processing is a convenient way to configure an application according to user input using the World-Wide-Web communications model. But, form processing using a CGI is extremely complex, as will be seen below, requiring an application designer to learn and implement an unfamiliar interface. A CGI is therefore not a suitable interface for rapid development and prototyping of new GUI capabilities. Moreover, a developer must then master a native application source code language (e.g., C, C++, etc.), HTML and the CGI, in order to develop a complete application along with its user interface.

Models of the World-Wide-Web communications paradigm for static content and dynamic content are shown in Figs. 14 and 15, respectively. As shown in Fig. 14, a browser 1401 makes a connection 1402 with a server 1403, which serves static content 1405 from a storage device 1407 to the browser 1401. In the case of dynamic content, shown in Fig. 15, the server 1403 passes control of the connection 1402 with the browser 1401 to an application 1501, through the CGI 1503. The application 1501 must maintain the connection 1402 with the browser 1401 and must pass control back to the server 1403 when service of the request which included dynamic content is complete. Furthermore, during service of a request which includes dynamic content, the application 1501 is responsible for functions normally performed by the server 1403, including maintaining the connection 1402 with the browser 1401, generating headers in the server/browser transport protocol, generating all of the static and dynamic content elements, and parsing any form data returned by the user. Since use of the CGI 1503 or other raw interface forces the application designer to do all of this work, applications 1501 to which forms are submitted are necessarily complex.

In order to provide dynamic content to a browser, the World-Wide-Web has also evolved to include Java and other client side scripting languages, as well as some server side scripting languages. However, these languages are interpreted by an interpreter built into the browser 1401 or server 1403, slowing down the presentation of information so generated. In the case of client side scripting, the script does not have any direct access to the application or to application specific information. Therefore, in order to generate or receive application specific information using client side scripting, the CGI 1503 or other raw interface must still be used. In the case of server side scripting, the server 1403 must parse the content as it is served, looking for a script to be interpreted. The access which a script has to the application is limited by the definition of the

scripting language, rather than by an application software interface designed by the application designer.

A server side script is an executable program, or a set of commands stored in a file, that can be run by a server program to produce an HTML document that is then returned to the Web browser. Typical script actions include running library routines or other applications to get information from a file, a database or a device, or initiating a request to get information from another machine, or retrieving a document corresponding to a selected hypertext link. A script may be run on the Web server when, for example, the end user selects a particular hypertext link in the Web browser, or submits an HTML form request. Scripts are usually written in an interpreted language such as Basic, Practical Extraction and Report Language (Perl) or Tool Control Language (Tcl) or one of the Unix operating system shell languages, but they also may be written in programming languages such as the "C" programming language and then compiled into an executable program. Programming in Tcl is described in more detail in Tcl and the Tk Toolkit, by John K. Ousterhout, Addison-Wesley, Reading, MA, USA, 1994. Perl is described in more detail in Programming Perl, by Larry Wall and Randal L. Schwartz, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1992.

Each document object in a web has an identifier called a Universal Resource Identifier (URI). These identifiers are described in more detail in T. Berners-Lee, "Universal Resource Identifiers in World-Wide-Web: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web," RFC 1630, CERN, June 1994; and T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994. A URI allows any object on the Internet to be referred to by name or address, such as in a link in an HTML document as shown above. There are two types of URIs: a Universal Resource Name (URN) and a Uniform Resource Locator (URL). A URN references an object by name within a given name space. The Internet community has not yet fully defined the syntax and usage of URNs. A URL references an object by defining an access algorithm using network protocols. An example URL is "http://www.agranat.com" A URL has the syntax "scheme: scheme_specific_components" where

"scheme" identifies the access protocol (such as HTTP, FTP or GOPHER). For a scheme of HTTP, the URL may be of the form "http://host:port/path?search" where

"host" is the Internet domain name of the machine that supports the protocol;

"port" is the transmission control protocol (TCP) port number of the appropriate server (if different from the default);

"path" is a scheme-specific identification of the object; and

"search" contains optional parameters for querying the content of the object.

URLs are also used by web servers and browsers on private computer systems or networks and not just the World-Wide-Web.

A site, i.e. an organization having a computer connected to a network, that wishes to make documents available to network users is called a "Web site" and must run a "Web server" program to provide access to the documents. A Web server program is a computer program that allows a computer on the network to make documents available to the rest of the World-Wide-Web or a private web. The documents are often hypertext documents in the HTML language, but may be other types of document objects as well, as well as images, audio and video information. The information that is managed by the Web server includes hypertext documents that are stored on the server or are dynamically generated by scripts on the Web server. Several Web server software packages exist, such as the Conseil Europeen pour la Recherche Nucleaire (CERN, the European Laboratory for Particle Physics) server or the National Center for Supercomputing Applications (NCSA) server. Web servers have been implemented for several different platforms, including the Sun Sparc 11 workstation running the Unix operating system, and personal computers with the Intel Pentium processor running the Microsoft® MS-DOS operating system and the Microsoft® Windows™ operating environment.

Web servers also have a standard interface for running external programs, called the Common Gateway Interface (CGI). CGI is described in more detail in How To Set Up And Maintain A Web Site, by Lincoln D. Stein, Addison-Wesley, August 1995. A gateway is a program that handles incoming information requests and returns the appropriate document or generates a document dynamically. For example, a gateway might receive queries, look up the answer in an SQL database, and translate the response into a page of HTML so that the server can send the result to the client. A gateway program may be written in a language such as "C" or in a scripting language such as Perl or Tcl or one of the Unix operating system shell languages. The CGI standard specifies how the script or application receives input and parameters, and specifies how any output should be formatted and returned to the server.

A user (typically using a machine other than the machine used by the Web server) that wishes to access documents available on the network at a Web site must run a client program

called a "Web browser." The browser program allows the user to retrieve and display documents from Web servers. Some of the popular Web browser programs are: the Navigator browser from NetScape Communications Corp., of Mountain View, California; the Mosaic browser from the National Center for Supercomputing Applications (NCSA); the WinWeb browser, from Microelectronics and Computer Technology Corp. of Austin, Texas; and the Internet Explorer, from Microsoft Corporation of Redmond, Washington. Browsers exist for many platforms, including personal computers with the Intel Pentium processor running the Microsoft® MS-DOS operating system and the Microsoft® Windows™ environment, and Apple Macintosh personal computers.

The Web server and the Web browser communicate using the Hypertext Transfer Protocol (HTTP) message protocol and the underlying transmission control protocol/internet protocol (TCP/IP) data transport protocol of the Internet. HTTP is described in Hypertext Transfer Protocol - HTTP/1.0, by T. Berners-Lee, R. T. Fielding, H. Frystyk Nielsen, Internet Draft Document, October 14, 1995, and is currently in the standardization process. At this writing, the latest version is found in RFC Z068 which is a draft definition of HTTP/1.1. In HTTP, the Web browser establishes a connection to a Web server and sends an HTTP request message to the server. In response to an HTTP request message, the Web server checks for authorization, performs any requested action and returns an HTTP response message containing an HTML document resulting from the requested action, or an error message. The returned HTML document may simply be a file stored on the Web server, or it may be created dynamically using a script called in response to the HTTP request message. For instance, to retrieve a document, a Web browser sends an HTTP request message to the indicated Web server, requesting a document by its URL. The Web server then retrieves the document and returns it in an HTTP response message to the Web browser. If the document has hypertext links, then the user may again select a link to request that a new document be retrieved and displayed. As another example, a user may fill in a form requesting a database search, the Web browser will send an HTTP request message to the Web server including the name of the database to be searched and the search parameters and the URL of the search script. The Web server calls a program or script, passing in the search parameters. The program examines the parameters and attempts to answer the query, perhaps by sending a query to a database interface. When the program receives the results of the query, it constructs an HTML document that is

returned to the Web server, which then sends it to the Web browser in an HTTP response message.

Request messages in HTTP contain a "method name" indicating the type of action to be performed by the server, a URL indicating a target object (either document or script) on the Web server, and other control information. Response messages contain a status line, server information, and possible data content. The Multipurpose Internet Mail Extensions (MIME) are a standardized way for describing the content of messages that are passed over a network. HTTP request and response messages use MIME header lines to indicate the format of the message. MIME is described in more detail in MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies, Internet RFC 1341, June 1992.

The request methods defined in the HTTP/1.1 protocol include GET, POST, PUT, HEAD, DELETE, LINK, and UNLINK. PUT, DELETE, LINK and UNLINK are less commonly used. The request methods expected to be defined in the final version of the HTTP/1.1 protocol include GET, POST, PUT, HEAD, DELETE, OPTIONS and TRACE. DELETE, PUT, OPTIONS and TRACE are expected to be less commonly used. All of the methods are described in more detail in the HTTP/1.0 and HTTP/1.1 specifications cited above.

Finally, a device or application using conventional World-Wide-Web technology must have access to a server. Conventional servers are large software packages which run on relatively large, resource-rich computer systems. These systems are resource-rich in terms of processing speed and power, long-term storage capacity, short-term storage capacity and operating system facilities. Conventional servers take advantage of these resources, for example, in how they store content source documents. For high-speed, convenient access to content, it is conventionally stored in a directory tree of bulky ASCII text files. Therefore, conventional World-Wide-Web technology cannot be used to implement a GUI in a relatively small, inexpensive, resource-poor device or application.

The combination of the Web server and Web browser communicating using an HTTP protocol over a computer network is referred to herein as the World-Wide-Web communications paradigm.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an improved graphical user interface (GUI) for use in connection with remote control, management, configuration, monitoring and diagnosing functions embedded in applications, devices and equipment.

According to one aspect of the invention, there is provided a method for providing a graphical user interface having dynamic elements. The method begins by defining elements of the graphical user interface in at least one text document written in a mark-up language. Next, the method defines including at a location in the document a code tag containing a segment of application source code. The text document is then served to a client which interprets the mark-up language; and when the location is encountered, the client is served a sequence of characters derived from a result of executing a sequence of instructions represented by the segment of application source code. An embodiment of code tags illustrating their use is described in detail, later.

According to another aspect of the invention, there is another method for providing a graphical user interface having dynamic elements. This method also defines elements of the graphical user interface in at least one text document written in a mark-up language. Included in the document is a string identified by prototype tags. The text document is served to a prototyping client which interprets the mark-up language but does not recognize and does not display the prototype tag, but does display the string. An embodiment of prototype tags illustrating their use is described in detail, later.

According to yet another aspect of the invention, there is yet another method for providing a graphical user interface having dynamic elements. Elements of the graphical user interface are defined in at least one text document written in a mark-up language. Included at a location in the document is a code tag containing a segment of application source code. Also included in the document is a string identified by prototype tags. The text document is compiled into a content source, which is subsequently decompiled into a replica of the text document. The replica of the text document is served to a client which interprets the mark-up language; and when the location is encountered in the replica, the client is served a character stream generated by executing the segment of application source code.

Yet another aspect of the invention is a software product recorded on a medium. The software product includes a mark-up language compiler which can compile a mark-up language document into a data structure in a native application programming language, the compiler

recognizing one or more code tags which designate included text as a segment of application source code to be saved in a file for compilation by a compiler of the native application programming language.

Another aspect of the invention is a method for providing a graphical user interface having displayed forms for entry of data. The steps of this method include defining elements of the graphical user interface in at least one text document written in a mark-up language; naming in the document a data item requested of a user and used by an application written in a native application programming language; and compiling the text document into a content source including a data structure definition in the native application programming language for the named data item.

Yet another aspect of the invention may be practiced in a computer-based apparatus for developing a graphical user interface for an application, the apparatus including an editor which can manipulate a document written in a mark-up language and a viewer which can display a document written in the mark-up language. The apparatus further includes a mark-up language compiler which recognizes a code tag containing a source code fragment in a native application source code language, the code tag not otherwise part of the mark-up language, the compiler producing as an output a representation in the native application source code language of the document, including a copy of the source code fragment.

In accordance with another aspect of the invention, there is a method for developing and prototyping graphic user interfaces for an application. The method includes accessing an HTML file, encapsulating portions of said HTML and entering source code therein, producing a source module from said HTML with encapsulated portions, producing source code for a server, and cross compiling and linking said application, said source code module and said server thereby producing executable object code.

The invention, according to another aspect thereof, may be a data structure fixed in a computer readable medium, the data structure for use in a computer system including a client and a server in communication with each other. The data structure includes cross-compiled, stored and linked, HTML files with encapsulated portions containing executable code associated with said application, server code, and application code, wherein said executable code is run when the HTML file is served thereby providing real time dynamic data associated with said application.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings, in which like reference numerals denote like elements:

Fig. 1 is a block diagram of that aspect of the invention relating to development systems;

Fig. 2 is a block diagram of that aspect of the invention relating to an embedded system;

Fig. 3 is an HTML text fragment illustrating the use of an EMWEB_STRING tag;

Fig. 4 is another HTML text fragment illustrating another use of an EMWEB_STRING tag;

Fig. 5 is an HTML text fragment illustrating the use of an EMWEB_INCLUDE tag;

Fig. 6 is another HTML text fragment illustrating another use of an EMWEB_INCLUDE tag;

Fig. 7 is an HTML text fragment showing a use of the EMWEB_ITERATE attribute in connection with an EMWEB_STRING tag;

Fig. 8 is an HTML text fragment showing a use of the EMWEB_ITERATE attribute in connection with an EMWEB_INCLUDE tag;

Fig. 9 is an example of forms processing showing the relationship between the HTML source code for the form and the form output produced;

Fig. 10 is a block diagram of the data structure which defines the header for the data.dat archive file;

Fig. 11 is a state diagram of an embedded system illustrating dynamic content processing;

Fig. 12 is a state diagram of an embedded system illustrating forms processing;

Fig. 13 is a state diagram of an embedded system illustrating suspend/resume processing;

Fig. 14 is a block diagram illustrating conventional World-Wide-Web communication of static content between a server and a client; and

Fig. 15 is a block diagram illustrating conventional World-Wide-Web communication of dynamic content between a server and a client.

DETAILED DESCRIPTION

The present invention will be better understood upon reading the following detailed description in connection with the figures to which it refers.

Embodiments of various aspects of the invention are now described. First, a development environment is described in which application development and graphical user interface development are closely linked, yet require a low level of complexity compared to conventional

development of an application and GUI. Second, an operating environment is described in which the application, a server and GUI are tightly coupled, compact and flexible. In the described system a GUI having portability, low run-time resource requirements and using any of a wide variety of systems available to a user as a universal front end, i.e. the point of contact with the user is software with which the user is already familiar.

Development Environment

Fig. 1 illustrates a development environment according to one aspect of the invention. Not all components of the environment are shown, but those shown are identified in the following discussion.

Conventionally, an application development environment may include a source code editor, a compiler 101, a linker and a run-time environment in which to test and debug the application. It is expected that development environments in accordance with the invention include those components of a conventional development environment which a developer may find useful for developing an application. In the case of embedded applications, i.e., applications included within a device or larger application, the run-time environment includes the device or application in which the application is embedded, or a simulation or emulation thereof.

The compiler 101 takes source code 103 generated using the source code editor or from other sources and produces object code 105, which is later linked to form the executable image.

In addition to the conventional elements noted above, the described embodiment of a development environment according to the invention includes an HTML compiler 107 whose output 109 is in the source code language of the application under development. In addition, the development environment may include an HTML editor, an HTTP-compatible server for communicating with client software, i.e., browsers, and an HTTP-compatible browser.

The HTML editor is used to create and edit HTML documents 111 which define the look and feel of a GUI for the application. Numerous tools are now available for performing this task while requiring a minimal knowledge or no knowledge of HTML, for example, Microsoft® Front Page™. It is preferred that the HTML editor used permit entry of non-standard tags into the HTML document.

As will be seen in further detail, below, the server and browser are used to test a prototype GUI before it is fully integrated with the application or in the absence of the application. The browser should be capable of making a connection with the server using, for

example, a conventional connection protocol such as TCP/IP, as shown and described above in connection with Fig. 14. Other protocols or direct connections can also be used, as would be understood by those skilled in this art. While the browser and the server may be connected through a network such as the Internet, they need not be. For example, the server and client may run and connect on a single computer system.

Application development proceeds substantially in a conventional manner as known to software developers. The application development should include the design of a software interface through which data will be communicated into and out of the application. However, the software interface is not a GUI. Rather, the interface merely defines how other software can communicate with the application. For example, the interface may be a collection of function calls and global symbols which other software can use to communicate with the application. The application should be written in a high level language such as C, C++, etc. The application can be tested by compiling and linking it with prototype code that provides or receives information through the software interface, exercising those features of the application.

Meanwhile, a GUI for the application is designed as follows. The look and feel of the GUI are developed using the HTML editor, server and browser to create a set of content source documents 111 including at least one HTML document, which together define the look and feel of the GUI. This aspect of GUI development is conventional, proceeding as though the developer were developing a World-Wide-Web site.

At locations in one or more HTML documents where data obtained from the application is to be displayed, the author includes special tags, explained further below, which allow the HTML document to obtain from the application the required data, using the application software interface.

The content source documents 111 are stored conventionally in the form of one or more directory trees 113. The directory tree 113 containing the content which defines the GUI is then compiled using the HTML compiler 107, to produce an application source code language output 109 representing the content source documents in the directory tree. The source code elements 109 produced from the content source documents 111 in the directory tree 113, source code for an HTTP compatible server (not shown) and the application source code 103 are compiled into object code 105 and linked to form an executable image. The server may be supplied in the form of an object code library, ready for linking into the finished executable image. The executable image thus formed fully integrates the graphical user interface defined using familiar tools of

World-Wide-Web content development with the control and other functions defined using conventional application development tools.

In order to successfully perform the integration described above, the HTML compiler 107 of the described embodiment of the invention, the EmWeb™/compiler 107, recognizes a number of special extensions to HTML. The HTML extensions implemented by the EmWeb™/compiler 107, embodying aspects of the invention are described in detail in Table A, Section 3.2. Several of these extensions are described briefly here, to aid in understanding the invention.

The EMWEB_STRING tag is an extension of HTML used to encapsulate a fragment of source code in the HTML document. The source code will be executed by a system in which the application is embedded when the document is served to a browser (usually running on another system) and the location of the EMWEB_STRING tag is reached. The source code returns a character string that is inserted as is into the document at the location of the EMWEB_STRING tag. Examples of the use of the EMWEB_STRING tag are shown in Figs. 3 and 4.

In the example of Fig. 3, the EMWEB_STRING tag 301 first defines using "C=" a boundary character 303 used to define the end 305 of the included source code. Immediately following the boundary character definition is a fragment of C code 307 which returns a pointer to a string representing one of three fax states. When served by an embedded application, this example HTML produces the text "NetFax State:" followed by "Sending", "Receiving" or "Idle", depending on the value of the symbol GlobalFaxState.

The example of Fig. 4 shows the use of EMWEB_STRING to output typed data whose type is defined by an attribute, EMWEB_TYPE 401. The EmWeb™/compiler uses this attribute 401 to produce a source code output routine which converts the typed data found at the address returned 403 into a string for serving at the proper location in the document.

A similar function is performed by the HTML extension, the EMWEB_INCLUDE tag. Using this tag, standard parts of a GUI such as headers and footers common to multiple pages or windows of information need only be stored once. Header and footer files are referred to using the EMWEB_INCLUDE tag which inserts them at the location in each HTML content document where the tag is placed. In the described embodiment of the invention, the contents of the EMWEB_INCLUDE tag must resolve to a relative or absolute path name within the local directory tree of content. This can be done by specifying a local Universal Resource Locator (URL), which is how resources are located in the World-Wide-Web communications paradigm, or by including source code which returns a string representing such a local URL. An absolute

local URL takes the form "/path/filename", where "/path" is the full path from the root of the directory tree to the directory in which the file is located. A relative URL defines the location of a file relative to the directory in which the current, i.e., base, document is located and takes the form "path/filename". While the described embodiment requires resolution of the EMWEB_INCLUDE tag to a local URL, the invention is not so limited. In alternate embodiments, local and external URLs may be permitted or other limitations imposed. Examples of the use of the EMWEB_INCLUDE tag are shown in Figs. 5 and 6.

In the example of Fig. 5, a COMPONENT attribute 501 in an EMWEB_INCLUDE tag simply defines a local URL 503.

In the more elaborate example of Fig. 6, a fragment of source code 601 which produces a local URL 603 upon a defined condition 605 is used to generate a local URL at run time.

The results to be returned by an EMWEB_STRING or EMWEB_INCLUDE tag can also be built up iteratively using repeated calls to the included source code. This is done using the EMWEB_ITERATE attribute, yet another extension to HTML. Examples of the use of EMWEB_ITERATE are shown in Figs. 7 and 8.

Fig. 7 shows an example of the EMWEB_ITERATE attribute 701 used in connection with the EMWEB_STRING tag 703. The fragment of code 705 is executed repeatedly until a NULL is returned. Thus, this HTML repeatedly executes the C source code fragment to display the tray status of all trays in a system.

Similarly, in Fig. 8, EMWEB_INCLUDE 801 and EMWEB_ITERATE 803 are used to build a table of features for which content from other URLs 805 are to be displayed. When the table is complete, a NULL is returned 807, terminating the iterations.

Since the extensions to HTML described above allow the encapsulation of source code within an HTML document a mechanism with which to provide the encapsulated source code with required global definitions, header files, external declarations, etc. is also provided in the form of an EMWEB_HEAD tag. The EMWEB_HEAD tag specifies a source code component to be inserted in the source code output of the EmWeb™/compiler, outside of any defined function. Although it is preferred that the EMWEB_HEAD tag appears in the HTML file header, it may appear anywhere. The code generated by an EMWEB_HEAD tag is placed before any functions or other code defined within the HTML content source documents.

As indicated above, the GUI may be prototyped using a conventional server and browser (see Fig. 14) to preview the HTML documents comprising the GUI. Therefore, it may be useful

to provide static content with which to preview the page, at locations where dynamic content will appear during use, but which does not appear in the compiled document. For example, it may be useful to include a prototyping value for content which is otherwise provided using the EMWEB_STRING tag mechanism. Therefore, another extension to HTML recognized by the EmWeb™/compiler is the EMWEB_PROTO begin 309 and end 311 tags, as shown in Fig. 3. The EmWeb™/compiler removes these tags and everything between them when compiling the document, but the tags are ignored and the text between them is interpreted normally by a conventional browser viewing the HTML document either directly or via a conventional server. Conventional browsers recognize the tag due to its special syntax, e.g., being enclosed in "<" and ">", but are designed to ignore and not display any tag for which the browser does not have a definition. All EmWeb™/compiler HTML extensions are thus skipped over by conventional browsers. Thus, in the example of Fig. 3, the prototype page displays "NetFax State: Sending". Fig. 4 shows a similar use of EMWEB_PROTO tags.

Handling of HTML forms by the EmWeb™/compiler is now described in connection with Fig. 9. As seen in Fig. 9, an HTML form is defined substantially conventionally. Names used in the form are used in constructing symbol names used in the output source code produced by the EmWeb™/compiler. Therefore names should be valid symbol names in the source code language.

Each element of a form definition is translated by the EmWeb™/compiler into a part of a corresponding data structure defined for that form. Forms data is moved into and out of the application by changing values of items in the data structure.

Turning now to the example in Fig. 9, the relationship between the illustrated HTML form definition and the corresponding data structure is described. The form is given a unique name, using an EMWEB_NAME attribute in a FORM tag. The form name becomes part of the structure name, for easy reference and uniqueness. The form name will also be used to generate function names for functions which are called when the form is served and when the form is submitted.

The structure generated is itself composed of two structures. The first holds values of each dynamic element of the form. The second holds a status flag indicating the status of the contents of a corresponding value. Thus, in the example of Fig. 9, a structure to hold values and status for the sysName INPUT and the Logging SELECTion is created. The value of sysName is a character string, while Logging is an enumerated type.

Two function prototypes are also generated. The actions to be performed by these functions must be defined by the developer. The Serve function is called when the form is served and can be used to supply default values, for example. The Submit function is called when the form is submitted, to update values in the data structure, for example.

Currently, EmWeb™/compiler supports TEXT, PASSWORD, CHECKBOX, RADIO, IMAGE, HIDDEN, SUBMIT, RESET, SELECT and OPTION input fields. For detailed descriptions, see Table A, Section 3.2.5. In addition, the EmWeb™/compiler supports "typing" of TEXT input field data. That is, the EMWEB_TYPE attribute may be used to define a TEXT input field to contain various kinds of integers, a dotted IP address (i.e., an address of the form 000.000.000.000), various other address formats, etc. A mapping of EMWEB_TYPE values to C language types is formed in the table in Table A, Section 3.2.5.3.

The EmWeb™/compiler has been described in terms of a generic application source code language. The current commercial embodiment of the EmWeb™/compiler assumes the application source code language to be C or a superset thereof, e.g., C++. However, the functionality described can be generalized to any application source code language which may be preferred for a particular application purpose. However, in order to more fully understand how the EmWeb™/compiler and HTML extensions described above cooperate to permit integration of an HTML defined GUI with an application defined in an application source code, it will be assumed, without loss of generality, that the application source code language is C or a superset thereof.

The EmWeb™/compiler produces a set of output files including a data.dat file containing the fixed data of a content archive, a code.c file containing the generated source code portions of an archive including portions defined in EMWEB_STRING, EMWEB_INCLUDE and EMWEB_HEAD tags and other source code generated by the EmWeb™/compiler, as well as proto.h and stubs.c files containing the definitions of C functions used for forms processing. The structure of these files is now described in connection with the data structure illustrated in Fig. 10.

The content archive file data.dat has a header structure as illustrated in Fig. 10. The data structure is accessed through an archive header 1001 which is a table of offsets or pointers to other parts of the archive. For example, there is a pointer 1001a to a compression dictionary 1003 for archives which include compressed documents. There is also a pointer 1001b to a linked list of document headers 1005, 1007 and 1009. Each document header 1005, 1007 and

1009 is a table of offsets or pointers to various components of the document. For example, the document header includes a pointer 1005a to the URL 1011 to which the document corresponds. There is also a pointer 1005b to a field 1013 giving the Multipurpose Internet Mail Extension (MIME) type of the document. There are pointers 1005c and 1005d respectively to header nodes 1015 and document nodes 1017, explained further below. Finally, there is a pointer 1005e to a block of static compressed or uncompressed data 1019 representing the static portions of the document.

The static data does not include any EmWeb™ tags, i.e., the extensions to HTML discussed above and defined in detail in Table A. Rather, information concerning any EmWeb™ tags used in the document appears in the document nodes structure.

Each EmWeb™ tag employed in a document is represented in that document's document nodes structure as follows. The location of the EmWeb™ tag within an uncompressed data block or an uncompressed copy of a compressed data block is represented by an offset 1017a relative to the uncompressed data. The type of tag is indicated by a type flag 1017b. A node may include a flag which indicates any attributes associated with the tag represented. For example, a node for a tag of type EMWEB_STRING may include a flag indicating the attribute EMWEB_ITERATE. Finally, nodes include an index 1017c. In nodes defining form elements, the index holds a form number and element number uniquely identifying the element and form within the document. In nodes defining EMWEB_STRING tags, the index is a reference to the instance of source code which should be executed at that point. As such, the index may be evaluated in an expression of a "switch" statement in C, where each controlled statement of the "switch" statement is one source code fragment from one EMWEB_STRING instance. Alternatively, the index may be a pointer or index into a table of source code fragments from EMWEB_STRING tags, which have been encapsulated as private functions.

The data structure defined above provides a convenient way of transferring control as a document containing dynamic content is served. When a document is requested, the list of document nodes is obtained, to determine at what points control must be transferred to code segments which had been defined in the HTML source document. The document is then served using the data block defining the static elements of the document, until each document node is encountered. When each document node is encountered, control is transferred to the appropriate code segment. After the code segment completes execution, the static content which follows is served until the offset of the next document node is encountered.

Header nodes permit the storage of document meta information, not otherwise handled, such as content language, e.g., English, German, etc., cookie control, cache control or an e-tag giving a unique version number for a document, for example a 30-bit CRC value computed for the document. By avoiding having to put this information in the header of each document, significant space can be saved in the archive because not all documents require this information. Therefore, header nodes need only be stored for documents using this information.

The data structure which represents the archive of content used by the EmWeb™/compiler embodiment of the invention is defined by the C source code contained in Table B.

Run-time Environment

Aspects of the invention related to the run-time environment and server are embodied in the EmWeb™/server as described in detail in Table A, Section 4.

To a conventional browser implementing HTTP, the EmWeb™/server behaves conventionally. However, as shown in Fig. 2, the EmWeb™/server is fully integrated with the application and therefore has access to information about the application and device in which it is embedded.

Operation of the EmWeb™/server with respect to presentation of dynamic content is now described in connection with Fig. 11.

Before the operations shown in Fig. 11 commence, one or more archives are loaded by the server. When each archive is loaded, the server generates a hash table using the archive header data structure to make documents easy to locate using URLs.

First, the browser requests a document at a specified URL, using HTTP 1101. The EmWeb™/server acknowledges the request, in the conventional manner 1103. The EmWeb™/server then uses the hash table of the archive header to locate the document requested and begin serving static data from the document 1105. When a document node is encountered, for example denoting the presence of an EMWEB_STRING tag, then the server passes control to the code fragment 1107a of the application which had been included in the EMWEB_STRING tag 1107. When the code fragment completes execution and returns some dynamic data 1109, the EmWeb™/server then serves that dynamic data to the browser 1111. The EmWeb™/server then resumes serving any static data remaining in the document 1113. This process continues until the entire document, including all dynamic elements has been served.

Run-time serving and submission of forms is now described in connection with Fig. 12. A brief inspection of Fig. 12 will show that form service and submission proceeds along similar lines to those for serving dynamic content.

The browser first requests a URL using HTTP 1201. When, during service of the contents of the URL requested, a form is encountered, service of the form and any HTML-defined default values commences normally. The EmWeb™/server then makes a call to the application code 1203 to run a function 1203a which may substitute alternate default values 1205 with which to fill in the form. The document served then is made to include the default values defined by the static HTML as modified by the application software 1207. Later, when the user submits the form, the browser performs a POST to the URL using HTTP 1209. The form data is returned to the application by a call 1211 made by the EmWeb™/server to a function 1211 which inserts the data returned in the form into the data structure defined therefor within the application code. The response 1213 is then served back to the browser 1215.

Finally, it should be noted that there may be times when a request for dynamic content may require extended processing, unacceptably holding up or slowing down other operations performed by the application. In order to avoid such problems, the EmWeb™/server implements a suspend/resume protocol, as follows. The suspend/resume protocol exists within a context of a scheduler maintained and operated by the server. The scheduler includes a task list of scheduled server tasks to be performed.

Fig. 13 illustrates a situation where a browser requests a document containing an EMWEB_STRING tag whose processing is expected to interfere with other application operations. The initial HTTP request 1301 for the document is acknowledged 1303, conventionally. When the EMWEB_STRING tag is encountered, control transfers 1305a to the appropriate source code fragment 1305b in the application. The application then calls the suspend function 1307 of the EmWeb™/server and returns a dummy value 1309 to the function call generated at the EMWEB_STRING tag location. Calling the suspend function 1307 causes the scheduler to remove the EMWEB_STRING processing task from the task list. When the application has finally prepared the dynamic content required in the original function call, the application calls a resume function 1311 of the EmWeb™/server. Calling the resume function 1311 requeues the EMWEB-STRING processing task on the task list, as the current task. The EmWeb™/server responds by calling 1305c the function 1305d defined at the EMWEB_STRING tag again, this time immediately receiving a response from the application in

which the requested dynamic content 1313 is returned. The dynamic content is then served to the browser 1315.

The suspend/resume feature is particularly useful in distributed processing environments. If an embedded application is running on one processor of a distributed environment, but dynamic content can be requested which is obtained only from another processor or device in the distributed environment, then the use of suspend/resume can avoid lockups or degraded processing due to the need to obtain the dynamic content through a communication path of the distributed environment. Consider, for example, a distributed system including a control or management processor, and several communication devices. An embedded application running on the management processor can be queried for configuration data of any of the communication devices. Without suspend/resume, obtaining that data would tie up the communication path used by the management processor for control of the various communication devices, degrading performance.

The described embodiment of the invention illustrates several advantages thereof. For example, an embedded application can now have a GUI which is independent of either the application platform of that used to view the GUI. For example, the GUI can be operated through a Microsoft® Windows™ CE machine, Windows™ 3.x machine, Apple Macintosh, WebTV box, etc. running conventional browser software. Also, development of a GUI for an embedded application is greatly simplified. The look and feel is designed using conventional HTML design techniques, including straight-forward prototyping of the look and feel using a conventional client server system, using simple HTML extensions. Integration with the embedded application does not require the developer to learn or develop any special interface, but rather uses some HTML extensions to incorporate application source code directly into the HTML content. Yet another advantage in that the entire embedded application along with an HTTP-compatible server and the content to be served is reduced to a minimum of application source code, data structures for static data and data structures for dynamic data.

The present invention has now been described in connection with specific embodiments thereof. However, numerous modifications which are contemplated as falling within the scope of the present invention should now be apparent to those skilled in the art. For example, the invention is not limited to content whose source is HTML. Any mark up language could be used in the context of this invention. Alternatively, the content source could be raw text, which is particularly suitable for situations where the output of the user interface is also processed by one

or more automatic software text filters. Therefore, it is intended that the scope of the present invention be limited only by the properly construed scope of the claims appended hereto.

EmWeb™ Functional Specification

by

Ian Agranat

Ken Giusti

Scott Lawrence

6/20/97

Table Of Contents

1. Introduction

The explosive growth of the World-Wide-Web in recent years has created a new standard for the Graphical User Interface (GUI) of networked applications: the Web browser. Netscape®, Mosaic, and other commercial and public-domain Web browsers are ubiquitous and inexpensive.

1.1. Web-based Network Management

Network management is a natural application for the World-Wide-Web. Before the Web, network management required end-users to purchase expensive custom-built GUI applications that used the SNMP protocol to manage network-based devices. With World-Wide-Web technology, many of these applications can be replaced by economical Web browsers.

World-Wide-Web standards make it possible to migrate the "look-and-feel" intelligence of the device configuration GUI away from third-party and custom-built network management applications and into the managed device itself. This migration gives vendors control over the configuration "look-and-feel" of their products among both low-cost Web browsers and the high-end third-party network management applications of the future.

Web-based network management can greatly accelerate the time-to-market for products traditionally managed by SNMP. With SNMP-based management, new product features require significant engineering resources to design and implement proprietary MIBs and develop customized GUI applications to manage the new features. On the other hand, Web-based management GUI applications can be rapidly prototyped and implemented using simple HTML forms.

SNMP will continue to play an important role in high-end network management applications by utilizing standard MIBs to monitor the overall topology and health of devices throughout the enterprise network. However, the new World-Wide-Web standards offer an exciting alternative to SNMP for the configuration and management of individual devices.

1.2. Web-based Systems, Devices, and Software

Embedded Web server technology can enable remote configuration, monitoring, and diagnostic capabilities for a wide range of applications. Imagine sending and receiving faxes, checking your voice-mail, and knowing the availability of your favorite candy bar in the vending machine down the hall by using your Web browser? Your office manager could configure the phone switch and voice-mail system, check the toner and paper levels in the photocopying machine, and monitor the security alarm system from their desktop. Your automobile mechanic could connect to your car's computer to perform engine diagnostics, and if they were stumped, the manufacturer could examine the results from around the world over the Internet.

1.3. Goals

EmWeb™ by Agranat Systems is a family of Embedded World-Wide-Web protocol software products engineered for the demanding real-time requirements of embedded systems.

Goals of the EmWeb product architecture include:

- Ease of use

  . Support rapid prototyping and implementation of new World Wide Web interfaces including device monitoring, configuration, and network management.
  . Easily portable to a wide-variety of real-time embedded operating environments with limited operating system capabilities, memory, and CPU resources.
  . Easily upgradable with new releases as World-Wide-Web standards continue to evolve rapidly.

- High performance

  . Efficient implementation for real-time processing environments.
  . Small foot-print to minimize FLASH and RAM resources required.

- Security

  . Support for authentication and security standards as they continue to evolve.
  . Configuration of method-independent access controls.

1.4. Assumptions

This functional specification assumes that the reader is familiar with the C programming language, the World-Wide-Web, and standards such as HTML (Hyper-Text Markup Language). Knowledge of HTTP (Hyper-Text Transport Protocol) and CGI (Common Gateway Interface) is also helpful to understand some of the more advanced features, but isn't required for most applications.

2. Overview

One draw-back to traditional World-Wide-Web servers is the CGI interface. Although HTML provides a convenient mechanism for sending interactive forms to a Web browser complete with text fields, checkboxes, and pull-down menus, the CGI interface used by server applications to process submitted forms is difficult to master. Form processing is essential for network management applications because it is the best way to implement a device configuration screen using World Wide Web protocols. However, CGI is not an appropriate interface for the rapid implementation and prototyping of new network management capabilities (EmWeb's number one goal). Another draw-back to traditional World-Wide-Web servers is the static nature of the content of files served to the Web browser. These files are typically resident on disk and are not expected to change frequently. On the other hand, the data of interest to network management applications is constantly changing (e.g. the number of packets received on a particular Ethernet port).

The EmWeb architecture works around the draw-backs of traditional World-Wide-Web servers by extending the standard HTML specification with new proprietary tags developed by Agranat Systems. These tags make it possible to provide a more intuitive HTML form processing interface for embedded systems, and give the application run-time control over the document content for the presentation of dynamic data. Note that these tags are not actually sent to the Web browser. Instead, they are stripped out and interpreted by the EmWeb/Compiler.

The EmWeb product consists of two components: The EmWeb/Compiler and the EmWeb/Server.

The EmWeb/Compiler is a tool provided by Agranat Systems that runs on an engineering development work-station under Unix, Windows/NT (x86), and Windows/95. The tool compiles one or more directories of files containing HTML, Java, text, graphics, etc., into one or more EmWeb archives. Each archive consists of an object code component and a fixed data component. These components are usually linked into the target's run-time image at compile time. However, the EmWeb architecture permits the dynamic loading and unloading of the fixed data components at run-time. Furthermore, if the target's operating system supports the dynamic loading and linking of object code modules, then entire archives may be loaded and unloaded at run-time.

The EmWeb/Server is a portable run-time library implemented in ANSI C which is compiled and linked with the target platform's application software. The EmWeb/Server implements the HTTP protocols to handle requests from Web browsers and interfaces with the archives created by the EmWeb/Compiler. The EmWeb/Server implements an application interface (API) to the target software environment's memory management, buffer management, and TCP/IP protocol stack.

3. EmWeb/Compiler

The EmWeb/Compiler processes directories of HTML and other Web documents to generate ANSI C code. The generated C code is then compiled using a C cross-compiler to generate object code for the target system. HTML forms are translated by the EmWeb/Compiler into a simple C function call interface customized to the application. The software developer is responsible for implementing the application-specific C code using these simple interfaces. The EmWeb/Compiler can generate function prototype definitions (a C include header file) to ensure correctness, and optionally generates stub functions to accelerate development.

The figure below illustrates the operation of the EmWeb/Compiler.

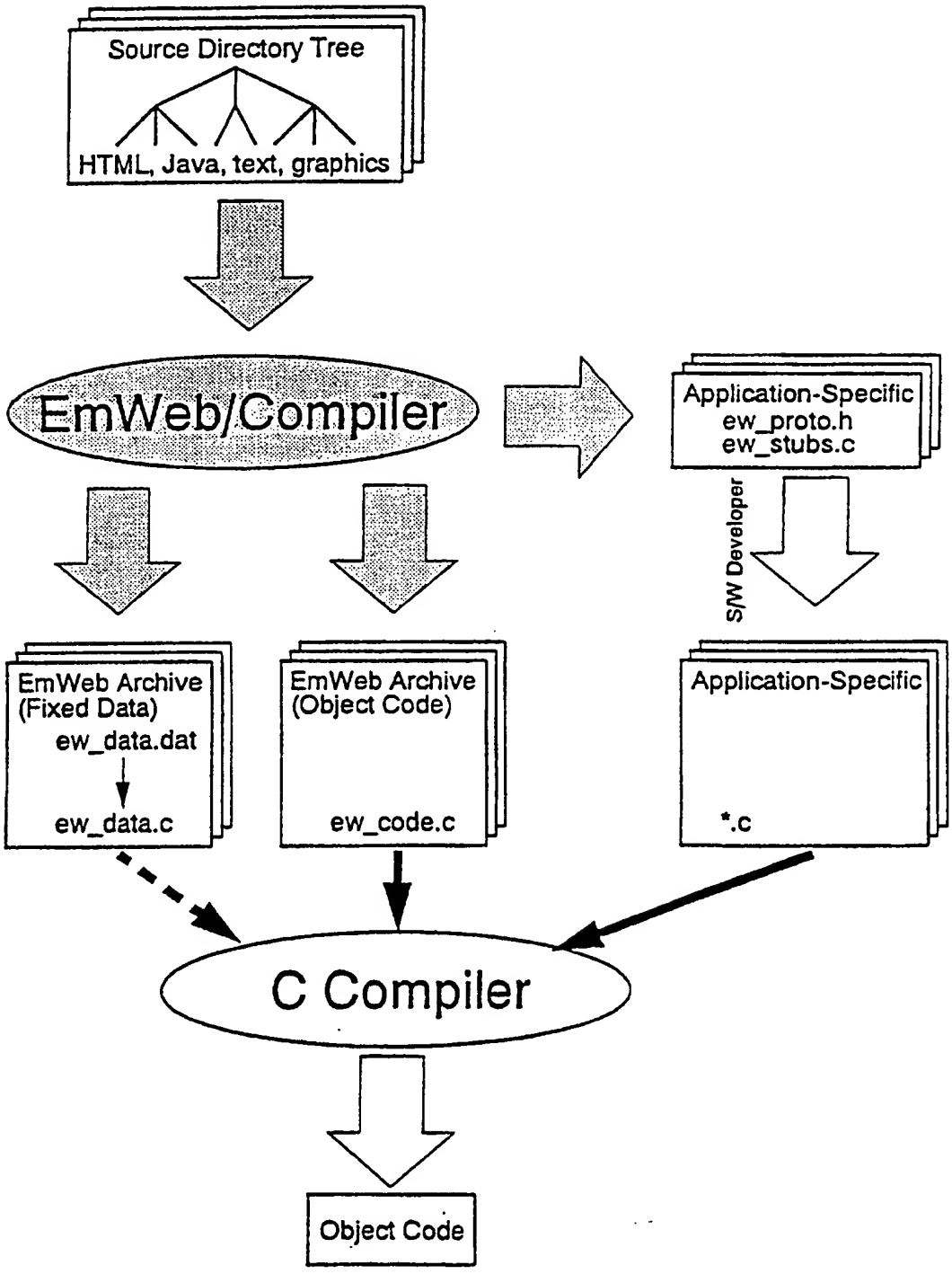


Figure 1: EmWeb/Compiler

3.1. EmWeb Context

The EmWeb/Server maintains a handle of type `EwsContext` at run-time that corresponds to a specific HTTP request from a Web browser client. This handle is made available to the application code responsible for providing the run-time content of HTML documents and forms.

The EmWeb/Server contains a library of functions available to the application for extracting information attached to this handle. For example, the application may want to determine the type of browser making a request, and change the look-and-feel of the requested document to take advantage of features available to particular browsers. In this example, the application could extract the browser type as follows:

```
bytes = ewsContextUserAgent (ewsContext,
        user_agent, sizeof(*user_agent));
```

For more information about available context information functions, refer to 4.1.5. Request Context Access, page 48.

3.2. HTML Extensions

The content to be provided by the EmWeb server is prepared as you would content for any normal Web server. The developer constructs a directory tree (or trees) of documents, which may include HTML documents, Java, text, graphics, and sound samples. The EmWeb compiler is then run over the content directory tree to build an internal database that is compiled and linked with the EmWeb/Server and the system code.

EmWeb defines proprietary HTML tags and attributes for access to system data and dynamic system control of the served documents. These tags are never served to browsers by the EmWeb/Server, and because (with one exception) they have no body part (they do not use a start and end tag, but only a stand-alone tag with attributes), they are ignored by browsers that see them when browsing the content tree either locally or via a non-EmWeb server. This facilitates testing and prototyping of content.

3.2.1. EMWEB_STRING

The `<EMWEB_STRING>` tag is used to provide applications with run-time control of an HTML document's content by encapsulating a fragment of C code in the document itself. This C code is executed as the document is being served to a Web browser and returns a null-terminated character string that is inserted as-is into the document in place of the `<EMWEB_STRING>` tag.

To include content in a document (which may include any standard HTML):

```
<EMWEB_STRING C='...; return (char *) somestring;'>
```

or

```
<EMWEB_STRING EMWEB_ITERATE C='...; r turn (char *) somestring;'>
```


The EmWeb/Compiler extracts the C language code fragment from the EMWEB_STRING tags in an HTML document and constructs a C module from them. Each fragment must end with a `return` statement that returns a pointer to a null-terminated string (or a NULL value which inserts no content). When the location of the EMWEB_STRING tag in the content is reached, the EmWeb/Server executes the code fragment and inserts the returned string into the document. If the optional EMWEB_ITERATE attribute is specified, the code fragment is called repeatedly until it returns a NULL pointer value. The number of iterations already processed (beginning with zero) is available to the code fragment by calling:

```
ewsContextIterations(ewsContext);
```

Note that the local variable

```
EwsContext ewsContext;
```

is made available to the EMWEB_STRING C code fragment by the EmWeb/Compiler.

The returned string must **not** be an automatic (stack) variable. The value is copied for transmission by EmWeb/Server immediately after returning. Thus, a single static string buffer may be shared by multiple code fragments. If pre-emptive multi-threading is to be used, the application may maintain a per-request buffer as part of its network handle. Alternatively, EMWEB_ITERATE may be used to support dynamic allocation of the string buffer as follows. On the first iteration (#0), memory may be allocated, filled with data, and returned to the EmWeb/Server for transmission. On the second iteration (#1), memory may be released and a NULL pointer returned to terminate the iteration.

The EmWeb/Compiler treats the embedded code opaquely; from the compiler's point of view, the code is simply a string of text that gets assembled into a generated code file. (See 3.6. Compiler Output, page 25).

As such, the EmWeb/Compiler needs to know the beginning and end of the embedded C code fragment. The character following the c= attribute in the EMWEB_STRING tag identifies the "bounds" of the embedded C code. This character must be either a single quote (') or double quote ("). When the EmWeb/Compiler identifies this "boundary character", it then treats all following characters as part of the embedded C code fragment, until it finds the next terminating "boundary character".

The obvious problem here is that the embedded C code can contain a single or double quote as part of the C code. For example:

```
return "Example of EmWeb's dynamic content\n";
```

In order to keep the EmWeb compiler from mistakenly confusing a single or double quote that belongs to the C code fragment as the ending "boundary character", the programmer must escape (using '\') any character intended for the C code that would be confused with the terminating boundary character. Thus, the above code fragment

could be embedded in an EMWEB_STRING tag in either of the following ways:

```
<EMWEB_STRING C='return "Example of EmWeb\'s dynamic content\n";'>
<EMWEB_STRING C="return \"Example of EmWeb's dynamic content\n\";">
```

The <EMWEB_STRING> tag may optionally be typed using the EMWEB_TYPE attribute as follows:

```
<EMWEB_STRING EMWEB_TYPE=DECIMAL_UINT C='
        return (uint32 *) &some_integer;'>
```

If EMWEB_TYPE is specified, then the C code fragment must return a pointer to data with the corresponding type. EmWeb/Server will convert the value into a character string automatically. A table of supported types can be found in section 3.2.5.3. HTML Form TEXT Input Fields, page 13

3.2.2. EMWEB_INCLUDE

The EmWeb/Server provides server-side inclusion of any part of its content tree using the EMWEB_INCLUDE tag; it is used in the same way as the EMWEB_STRING tag:

```
<EMWEB_INCLUDE COMPONENT='Local-URL'>
```

or

```
<EMWEB_INCLUDE C='...; return (char *) localURL;'>
```

or

```
<EMWEB_INCLUDE EMWEB_ITERATE C='...; return (char *) localURL;'>
```

If the COMPONENT tag is specified, the specified local element is resolved and inserted into the current document. This allows standard parts such as headers and footers to be stored only once.

As in EMWEB_STRING, the fragment specified by the C= attribute must end with a `return` statement that returns a pointer to a null-terminated string (or NULL). For EMWEB_INCLUDE, however, this string is treated as a relative or absolute path name within the local content tree. The path name is resolved to an entire document, which is then inserted in the current document for return to the browser. If the optional EMWEB_ITERATE attribute is specified, the code fragment is called repeatedly until it returns a NULL pointer value. The number of iterations already processed (beginning with zero) is available to the code fragment using the same access routine used for EMWEB_STRING. This tag is useful for building a response from components that may be part of the original document tree, or created dynamically.

Note that the local variable

```
EwsContext ewsContext;
```

is made available to the EMWEB_INCLUDE C code fragment by the EmWeb/Compiler.

The same quoting rules as EMWEB_STRING apply to EMWEB_INCLUDE.

3.2.3. EMWEB_HEAD

Because the <EMWEB_STRING> and <EMWEB_INCLUDE> tags allow the application to encapsulate fragments of C code within an HTML document, a mechanism is needed to provide the code fragments with the necessary global definitions, header files, external declarations, etc. needed for compilation. The <EMWEB_HEAD> tag is provided for this purpose.

```
<EMWEB_HEAD C='#include file-name.h'>
```

The EMWEB_HEAD tag, if present, should appear in the HEAD portion of the HTML document (this is not enforced, but the code fragments from all EMWEB_HEAD fragments are placed at the beginning of the generated module, so doing so is less confusing); it specifies a source fragment to be inserted in the generated module outside any C function. This fragment may declare variables, constants, or (as shown in the example above) C preprocessor directives - it could even define entire subroutines. This fragment is never executed directly by the EmWeb Server, so it does not insert any content into the document at the location of the tag.

Note that all EMWEB_HEAD code fragments from all HTML files in an archive are combined into the generated archive object code component source file. Therefore, it is recommended that common `#include` directives appear only in one document in the archive (typically, the index.html document at the archive's root directory).

The same quoting rules as EMWEB_STRING apply to EMWEB_HEAD.

3.2.4. EMWEB_PROTO

When designing an HTML document, it may occasionally be useful to provide content with which to preview the page, but which does not appear in the compiled document. This is accomplished with the EMWEB_PROTO begin and end tag; the EmWeb/Compiler removes these tags and everything between them when producing the document database, but the text between them is interpreted normally by a browser viewing the HTML document directly or via a conventional server. For example:

```
<H4>
System:  <EMWEB_STRING C="return mib2sysName;">
         <EMWEB_PROTO>system-name</EMWEB_PROTO>
</H4>
```

When compiled, the system name displayed in the first level header will be the value in the string variable `mib2sysName`, but viewed with a browser when previewing it looks like:

```
System: system-name
```

Note that the EmWeb/Compiler ignores HTML comments (e.g. "<!-- ... -->"). If EMWEB_STRING and EMWEB_INCLUDE tags are present within HTML comments, they will

be expanded by EmWeb/Server. If it is desirable to "comment out" an EMWEB_STRING or EMWEB_INCLUDE tag, EMWEB_PROTO tags should be used instead.

3.2.5. Forms

Form processing takes place as two operations: serving the form to the browser, possibly with current or default values set in some form input elements, and processing the values from the submitted form. Each of these operations is assisted by an application-provided routine associated with the form. Prototypes for these two application routines are produced for each form by the EmWeb/Compiler using information provided by EmWeb attributes on the FORM and its input elements.

> ***Restriction:*** *While HTML allows it, EmWeb does not support nested* FORM *tags (you may not have a form within a form).*

The elements for each form are combined in a generated C data structure by the EmWeb Compiler; this structure is passed by the EmWeb Server to each of the two form processing routines, and includes information on the status of each element.

3.2.5.1. Form Element Names

The EMWEB_NAME attribute is used to specify the names to be used by the EmWeb/Compiler when generating function and structure names.

On the FORM tag, the EMWEB_NAME attribute specifies a base name to be used to generate the function names and the form data structure name. (This implies that the value of the EMWEB_NAME attribute must be a valid C identifier). Note that if EMWEB_NAME is not present in a <FORM> tag, then the form is simply ignored by the EmWeb/Compiler and is passed as-is to the browser.

Consider the following example. Suppose there exists an HTML document "example.html" that contains:

```
<FORM METHOD=POST EMWEB_NAME=foo ACTION=bar >
```

The ACTION attribute is optional; if ACTION is not specified, then there may be only one form in the document. If ACTION is specified, it is served to the browser as "ACTION=example.html/bar". (This allows EmWeb/Server to differentiate between multiple forms in a document. It also allows the submission of forms to be protected by different access controls than the serving of forms. See 3.4. _ACCESS Files, page 22).

The EmWeb/Compiler generates the following function and structure prototypes:

```
typedef struct EwaForm_foo_s
 {
   struct
     {
       ...
     } value;
```



```
    struct
      {
        ...
      } status;
  } EwaForm_foo;...

void ewaFormServe_foo (EwsContext context, EwaForm_foo *formp);
char * ewaFormSubmit_foo (EwsContext context, EwaForm_foo *formp);
```

The ellipsis (...) above are replaced by declarations of the structure members to hold the data for each form element. The standard HTML NAME attribute is used in each INPUT tag of the form to specify the structure member names within the form data structure.

> ***Restriction:*** *While HTML allows arbitrary string values for NAME attributes, EmWeb requires that NAME attributes are valid C identifiers since they are used as field names in the form's C structure definition.*

The application-provided `ewaFormServe_foo` function is invoked by EmWeb/Server when the HTML document containing the form is requested by a Web browser. This provides the application with an opportunity to modify the default values displayed by the browser at run-time.

The application-provided `ewaFormSubmit_foo` function is invoked by EmWeb/Server when the corresponding HTML form is submitted. This provides the application with an opportunity to process the submitted form data and generate an appropriate response.

3.2.5.2. Form Data Structure

The generated form data structure passes information for the form between the system and the EmWeb/Server. It has two sections: 'value' and 'status', each containing one or more members for each form element. The 'status' section member is a flag that indicates the status of the contents of the 'value' member. The 'value' member contains the data for the form element in an internal representation which may be specified by the form author using the EMWEB_TYPE attribute. For example:

```
<INPUT TYPE=TEXT SIZE=15 MAXSIZE=15 VALUE=127.0.0.1
 NAME=ifAddr EMWEB_TYPE=DOTTED_IP>
```

The example above produces the following structure members:

```
typedef struct EwaForm_foo_s
 {
   struct
     {
       ...
       uint32     ifAddr;
```



```
        uintf       ifAddr_size;
        uintf       ifAddr_maxlength;
        ...
      } value;
    struct
      {
        ...
        uint8       ifAddr;
        ...
      } status;
  } EwaForm_foo;
```

The `value.ifAddr` member contains the host-order 32-bit IP address. The `ewaFormServe_foo` function sets this structure member with the value to be displayed on the form and sets `status.ifAddr` to `EW_FORM_INITIALIZED`. (Note that if `status.ifAddr` is not changed by the application, then the form is displayed with the value specified by the `VALUE` tag, "`127.0.0.1`"). When the Web browser end-user has entered a valid IP address and submits the form, the EmWeb/Server sets `value.ifAddr` to the value provided by the browser and sets `status.ifAddr` to `EW_FORM_RETURNED` before invoking `ewaFormSubmit_foo`.

The `value.ifAddr_size` member is generated because the `SIZE` attribute was specified and defaults to 15. This gives `ewaFormServe_foo` an opportunity to override the field size value before serving the form. (This field is not used by `ewaFormSubmit_foo`).

The `value.ifAddr_maxlength` member is generated because the `MAXSIZE` attribute was specified and defaults to 15. This gives `ewaFormServe_foo` an opportunity to override the maximum field size value at run-time. (This field is not used by `ewaFormSubmit_foo`).

The `status.ifAddr` member is used to indicate the status of the form element member and consists of a bit field defined as follows (from `src/include/ews_def.h`):

```
/*
 * For ewaFormServe_*
 */
#define EW_FORM_INITIALIZED 0x01    /* set if field initialized */
#define EW_FORM_DYNAMIC     0x02    /* set if field ewaAlloc'ed */

 /*
  * For ewaFormSubmit_*
  */
#define EW_FORM_RETURNED    0x10    /* set if value returned */
#define EW_FORM_PARSE_ERROR 0x20    /* set if value invalid */
#define EW_FORM_FILE_ERROR 0x20     /*        or i/o error */
```

For the `ewaFormServe_*` call (when the form is being sent to the browser), the structure is passed with the status for each element set to either zero, or `EW_FORM_INITIALIZED` if a default value was specified by the HTML `VAULE` tag. When

control is returned to the EmWeb/Server, the server checks each status and uses the new value for each field for which the EW_FORM_INITIALIZED is set. Initialized values override any VALUE, CHECKED, or SELECTED values specified in the source HTML document. The EW_FORM_DYNAMIC flag is only valid for type fields containing pointers to data (including TEXT, PASSWORD, HIDDEN, TEXTAREA, SUBMIT, and EMWEB_TYPE=HEXSTRING) and indicates that the memory referenced by the pointer was allocated by ewaAlloc and must be freed by EmWeb/Server using ewaFree after the form is served.

When the form is submitted by the browser, the server attempts to convert each value returned by the browser to the appropriate type. The server sets the status member to indicate the status of the data member of the structure, and then calls the ewaFormSubmit_* routine associated with the form. The EW_FORM_RETURNED flag is set if a valid value was returned for the field in the submission. The EW_FORM_PARSE_ERROR flag is set if a value was returned, but was not parsed correctly for the specified EMWEB_TYPE.

> *Note: When a form is submitted, the contents of a value field is undefined unless the EW_FORM_RETURNED flag is set in the corresponding status byte.*

The application processes the submitted values and returns a string containing the URL to use as the redirection response to the Web browser, or NULL for an empty ("204 No Content") response. Alternatively, the following function may be invoked to return a local relative URL from the archive as a response.

```
EwsStatus ewsContextSendReply ( EwsContext context, char * url );
```

3.2.5.3. HTML Form TEXT Input Fields

A text field can be used to represent a wide variety of value types. For example, a text field may be used to configure an IP address in which case the natural C-language representation is a host-order 32-bit integer. By default, a text field is simply a null-terminated character string. However, the EMWEB_TYPE attribute may be used to indicate any of a number of supported EmWeb types. Refer to the table below for details.

HTML text fields may specify a field size and/or a maximum length. If so, additional _size and _maxlength structure members are added to the form structure to give the application run-time control over these values. The format and resulting form structure fields of a text input field is given below.

```
<INPUT TYPE=TEXT NAME=name [MAXLENGTH=#max] [SIZE=#size] [VALUE=value]
[EMWEB_TYPE=ewtype]>

EwType   name;              /* defaults to value */
uintf    name_maxlength;    /* defaults to #max */
uintf    name_size;         /* defaults to #size */
```

enumerated type's integer value to its name as specified by the VALUE attribute:

```
const char *ewsFormEnumToString ( EwsContext context, int value );
```

3.2.5.7. HTML Form IMAGE Input Fields

Images are "read only" from the point of view of the application as they return the x and y image coordinates selected by the end-user from the Web browser. Therefore, the generated structure value members are only useful in the ewaFormSubmit_* application function. Structure members are generated representing the x and y coordinates as follows.

```
<INPUT TYPE=IMAGE NAME=name SRC=src [ALIGN=align]>

typedef struct EwaForm_foo_s
{
  struct
    {
      ...
      uint32  name_x;
      uint32  name_y;
      ...
    } value;
  struct
    {
      ...
      uint8   name;
      ...
    } status;
} EwaForm_foo;
```

3.2.5.8. HTML Form HIDDEN Input Fields

Hidden fields are identical to text fields except that they are not displayed (and thus not modified) by the browser. They are typically used by applications to pass context information to the browse. This context information is returned to the application during submission.

```
<INPUT TYPE=HIDDEN NAME=name [VALUE=value] [EMWEB_TYPE=ewtype] >

EwType name;        /* defaults to value if present */
```

3.2.5.9. HTML Form SUBMIT Input Fields

Submit buttons with NAME attributes are represented by a null-terminated character string. Submit buttons without NAME attributes are not included in the form structure.

```
<INPUT TYPE=SUBMIT [NAME=name] [VALUE=value] >

char *name;         /* defaults to value if pres nt */
```

3.2.5.10. HTML Form RESET Input Fields

No structure member is generated for a reset button.

3.2.5.11. HTML Form SELECT/OPTION Fields

Select boxes may be used to select a single item from a list (the default), or multiple items from a list (if the attribute MULTIPLE is present). For single item selection, values are represented by a C enumerated type generated by the EmWeb/Compiler. (The enumerated types defined by radio buttons and single select options are combined into a single archive-wide enumeration).

```
<SELECT NAME=name {SIZE=#size} >
        <OPTION VALUE=value1 {SELECTED} >
        <OPTION VALUE=value2 {SELECTED} >
</SELECT>

typedef enum EwaFormEnum_ew_archive_e
 {
   ...
  ,value1
  ,value2
   ...
 } EwaFormEnum_ew_archive;

EwaFormEnum_ew_archive name;/* defaults to selected value */
```

For multiple item selection, a status and value form structure field is generated for each option as follows:

```
<SELECT NAME=name {SIZE=#size} MULTIPLE >
        <OPTION VALUE=value1 {SELECTED} > descriptive text
        <OPTION VALUE=value2 {SELECTED} > descriptive text
</SELECT>

boolean value1;              /* defaults to TRUE if SELECTED */
boolean value2;              /* defaults to TRUE if SELECTED */
uintf name_size;             /* defaults to #size */
```

HTML select fields may specify the size of the selection box. If so, an additional _size structure member is added to the form structure to give the application run-time control over this value.

> ***Restriction:*** *While HTML allows arbitrary string values for* VALUE *attributes, EmWeb requires that* VALUE *attributes for select options are valid C identifiers since they are used as enumerated type or structure field identifiers.*

3.2.5.12. TEXTAREA

The textarea value is a null-terminated string of characters. Additional value fields are placed in the form structure for the number of rows and columns in the textarea.

These fields are initialized by EmWeb/Server to the defaults specified in the source HTML document, but may be overridden at run-time by the application's `ewaFormServe_*` function.

```
<TEXTAREA COLS=#cols ROWS=#rows NAME=name >
...
</TEXTAREA>

char     *name;          /* null-terminated string */
uintf    name_cols;      /* default to #cols */
uintf    name_rows;      /* default to #rows */
```

Note that the `*name` points to the text present in the TEXTAREA input field.

3.2.5.13. File Upload

RFC1867 file upload defines a new form input type as follows:

```
<INPUT TYPE=FILE NAME=name [VALUE=value] [SIZE=#rows[,#cols]]>
```

> Note: Use of file input types requires that the form uses `multipart/form-data` as an encapsulation type. This must be specified in the <FORM> tag as follows:
>
> ```
> <FORM METHOD=POST ENCTYPE=multipart/form-data ... >
> ```

This causes the browser to prompt the user for a local (relative to the browser) filename (by default, value is used if specified). On submission, the file is uploaded to the server along with other form field values.

EmWeb/Compiler parses file type input fields into the following fields in the form value substructure:

```
char             *name;          /* serve only: default filename */
EwaFileHandle    name_handle;    /* submit only: received file */
uintf            name_size;      /* serve only: #rows (no #cols) */
uintf            name_rows;      /* serve only: #rows (w/#cols) */
uintf            name_cols;      /* serve only: #cols */
```

In addition, the following fields in the form status substructure are generated as well:

```
uint8            name;           /* status for filename */
uint8            name_handle;    /* status for received file */
```

For more detailed information on the file upload interface, please refer to 4.1.8. Local Filesystem Interfaces, page 54.

3.2.6. Graphic Maps

The HTML ISMAP attribute on an IMG tag specifies that the file is an image map. Such an IMG tag must always be enclosed within an anchor that specifies a hyper-link URL

The ISMAP attribute causes a click within the image to be sent as a request to the URL in the enclosing anchor, qualified by the x & y coordinates of the click within the image.

EmWeb provides support for image maps through a map element in the HTML source tree which specifies rectangular regions and URLs to be associated with them.

The interface designer creates an image map file, a text file which defines the URLs to be provided for each region of the image. The syntax of the entries is:

```
rect-line | default-line

rect-line          ::= rect url point point

point              ::= x,y

default-line       ::= default url
```

The `rect` token specifies a URL to be served in response to the specified top-left and bottom-right coordinates. The optional `default` specifier provides the URL for any coordinate not specified by some `rect` (only one `default` is allowed). Order of the `rect` lines is significant - if the regions overlap the first match takes precedence.

If no match is found, and no default is specified, the server returns "no content" to the browser, effectively a no-operation. Otherwise, the matching URL is returned to the browser as a redirection. The URL may be relative to the map file in the archive, or absolute.

The map files are compiled by EmWeb/Compiler so that they do not need to be parsed at run-time by the EmWeb/Server. The following illustrates the syntax of a typical map file:

```
rect example1.html 15,18 169,37
rect example2.html 157,46 385,65
rect example3.html 365,71 460,87
default /home.html
```

If the above file was "`sample.map`", and corresponded to the image "`/pictures/sample.gif`", the following HTML would be used to implement the map:

```
<A HREF="sample.map"> <IMG SRC="/pictures/sample.gif" ISMAP> </A>
```

By convention, map files use the suffix '`.map`', but this can be changed. See 3.3. mime.types Configuration File, page 20, and 3.4. _ACCESS Files, page 22.

3.2.7. Cache Control

The HTTP/1.0 protocol has some simple provisions for communicating cachability information to caches and proxies between the server and the browser. HTTP/1.1 provides even greater cache control features.

By default, EmWeb/Server classifies compiled-in documents as being either static or dynamic. A document is considered dynamic if it contains one or more EMWEB_STRINGS, EMWEB_INCLUDES, or EmWeb-enhanced HTML forms. Otherwise, the document is static.

When serving a static document, EmWeb/Server sends a Last-Modified HTTP header indicating the time and date that the corresponding archive was created by the EmWeb/Compiler. No other cache control headers are generated in HTTP/1.0. With HTTP/1.1, a Cache-Control header is generated indicating either "public" if the document is not protected by an authentication realm, or "private" if it is.

When serving a dynamic document, EmWeb/Server sends a Last-Modified and Expires HTTP header indicating the current time and date, as well as "Pragma: no-cache" for HTTP/1.0 and "Cache-Control: no-cache" for HTTP/1.1.

It may be desirable to override the above default behavior for certain applications. Specifically, an application may desire static treatment of a dynamic document. For example, if EMWEB_INCLUDE tags are used to insert common headers and footers, EmWeb/Server would treat the document as dynamic even though the content may actually be static. The following EmWeb tag may be used anywhere in an HTML document to mark it as static for cache control purposes:

```
<EMWEB_STATIC>
```

3.2.8. EmWeb CGI

The mechanisms described above are designed to meet most needs of the system designer without having to deal with the complexities of direct access to the HTTP input and output mechanisms. If such access is desired, the EmWeb raw CGI interface provides access similar to that available in a CGI program environment on any HTTP server. See 4.1.6. Raw CGI, page 51.

3.3. mime.types Configuration File

The `mime.types` configuration file is read by EmWeb/Compiler before processing source directories of HTML and other Web documents. If "-m" is specified on the EmWeb/Compiler command line, then the specified file is used. Otherwise, EmWeb/Compiler searches for a `mime.types` file by first looking at the $EMWEB_MIME environment variable, and then looking at $EMWEB_HOME/$EMWEB_MIME. If the $EMWEB_MIME environment variable is not specified, it defaults to "`mime.types`". If the $EMWEB_HOME environment variable is not specified, it defaults to "`/usr/local/share/emweb`".

The `mime.types` file contains default parameters for files derived from the file name suffix. For example:

```
#
# EmWeb/Compiler mime.types file example
# Anything aft r a '#' is a comment.
# A line whose first character is white space is a continuation line
```



```
# Each specifier must  nd in ';'
#

.html    type=text/html parse=emweb_html compress ;
.txt     type=text/plain parse=emweb_text compress ;
.gif     type=image/gif parse=binary ;
.map     imagemap;
.        index=index.html;
```

The suffix may be any tail match; it is not restricted to values starting with '.'. The `mime.types` file contains specifiers for the default access rights and parameters of files ending with a particular suffix. Each specifier is the file name suffix, followed by one or more of the following attributes, followed by a semicolon:

type=*mime-type*

Specifies the MIME encapsulation media type represented by the file. Media type values are registered with the Internet Assigned Number Authority (IANA). Use of non-registered media types is discouraged. (Note that the *mime-type* must be quoted if it contains a ';' line-terminator character). The default type depends on the parser selected (see below).

parse=*emweb_html | emweb_text | text | binary*

Specifies how the EmWeb/Compiler is to parse the file.

EMWEB_HTML indicates that the file contains HTML and the EmWeb/Compiler handles <EMWEB_STRING>, <EMWEB_INCLUDE>, <EMWEB_PROTO>, <EMWEB_HEAD>, and EmWeb HTML forms. The default type for the EMWEB_HTML parser is "text/html".

EMWEB_TEXT indicates that the file contains text which may contain <EMWEB_STRING>, <EMWEB_INCLUDE>, <EMWEB_PROTO> and <EMWEB_HEAD> directives. However, such a file may not contain EmWeb HTML forms. The default type for the EMWEB_TEXT parser is "text/plain".

TEXT indicates that the file contains text which should not be parsed (e.g. PostScript). The default type for the TEXT parser is "text/plain".

BINARY indicates that the file contains raw binary data. The default type for the BINARY parser is "application/octet-stream".

The default parser is BINARY if not otherwise specified.

compress

Specifies that the content should be compressed. The EmWeb compression algorithm creates an archive-wide dictionary of common strings which are referenced by the individual documents. Compression ratios for text files of 50% can be achieved if there is sufficient redundancy

throughout the archive. However, compression ratios will vary widely among applications. The default is no compression.

nocompress
Specifies that the content should not be compressed. (This is typically used in an _ACCESS file to override a specification in `mime.types`). The default is no compression.

imagemap
Specifies that the content is an imagemap that defines the coordinates for an image. This implies "hidden", and inherits realm membership from the current directory.

ignore
Specifies that the entry should be skipped (i.e. not included in the archive). This is typically used to exclude source control directories, backup files from editors, etc.

The current directory is specified using '.' as the suffix name. Directories may contain the attribute:

index=*index-file-name*
The element (in the directory) to be returned in response to an access request to that directory (a URL whose last component is the directory name).

3.4. _ACCESS Files

Access to individual files and directories is controlled by a configuration file in each source archive directory named '_ACCESS'. The _ACCESS files contain specifiers for the access rights of elements in the corresponding directory (and optionally for the directory itself using the '.' specifier). Specifiers in the _ACCESS file override specifiers in the `mime.types` file. Each specifier is the file name (not a suffix) followed by any of the attributes defined above for `mime.types`. The following additional attributes are permitted in _ACCESS files.

realm=*realm-name*
requires that any request to access the file or directory must be authenticated in the named realm. If an empty realm is given (i.e. `realm=""`), then the file may be accessed without any authentication. See 4.1.4. Authentication and Security, page 39.

hidden
Specifies that the file is included for use in constructing dynamic content and may not be seen or accessed directly by a Web browser. (Note that the hidden and realm attributes are mutually exclusive).

link=*url*

Specifies that this element is linked to a different URL. When accessing this file, the browser shall be redirected to the specified relative or absolute URL. The link attribute may not appear with any other attributes.

If `realm` is specified for a directory, it applies as a default for all files and subdirectories within the current directory. Specifications for individual files in an `_ACCESS` file override defaults specified for the directory.

If `hidden` is specified for a directory, it applies as a default for all files and subdirectories within the current directory. Individual files and subdirectories may be overridden by specifying a `realm`.

Note that a default '.' specification for index may be included in the `mime.types` file and overridden in individual `_ACCESS` files.

In addition to the elements in the directory, the `_ACCESS` file may specify the base name for CGI elements (which do not need actual files in the HTML tree). These are specified with the access attribute cgi below and may also have a `realm` attribute.

cgi=*symbol*

The element is a CGI application. The symbol value is used by the EmWeb/Compiler to generate application-specific prototypes for the raw-CGI interface. See 4.1.6. Raw CGI, page 51.

Finally, form action URLs used when a browser submits an EmWeb HTML form to the EmWeb/Server may be protected by a specific realm which may be different from the realm of the document containing the form. In this way a form may be used to display data to a less restricted realm, while permitting changes only from users in a more restricted realm. For example, if the HTML file "`example.html`" contained an EmWeb HTML form tag `<FORM METHOD=POST EMWEB_NAME=foo ACTION=bar>`, the following specifier may appear in the `_ACCESS` file corresponding to the directory containing "`example.html`".

```
example.html/bar  realm=realm-name
```

Access control is only computed by the EmWeb/Server on the initial URL; if the element to be returned in response to the URL includes an `EMWEB_INCLUDE` tag, the EmWeb/Server does not perform an access check on those individual elements.

3.5. Compiler Options

The EmWeb/Compiler program "ewc" takes the following command-line arguments followed by a list of one or more directories and/or files with which to build an EmWeb archive.

(-n | --name) <archive-symbol-base-name>

The base symbol name produced by EmWeb/Compiler corresponding to the object part (and, if -c is also specified, the data part) of the archive. By default, this parameter is "ew_archive". Thus, the following symbol is generated by the EmWeb/Compiler in ew_code.c:

```
EwsArchive ew_archive;
```

And if the -c option is specified, the following symbol is generated in ew_data.c.

```
uint8 ew_archive_data[];
```

(-m | --mime) <mime.types filename>

The path name to the mime.types configuration file. By default, EmWeb/Compiler looks in $EMWEB_MIME (*mime.types*) followed by $EMWEB_HOME/$EMWEB_MIME (*/usr/local/share/emweb/mime.types*).

(-u | --url) <archive-url-base-path>

The base URL path of the generated archive. By default, this is '/'.

(-r | --raw) | (-c | --c)

The output format of the generated archive data component (raw by default). If --raw is specified, the data component of the archive is written to the file ew_data.dat as a raw binary file. If --c is specified, the data component of the archive is written to the file ew_data.c as a C file that, when compiled, produces an array of data reference by the symbol ew_archive_data. Both flags may be specified on the command line to cause the generation of both raw and C output files.

(-o | --output)

The prefix for generated files data.dat, data.c, code.c, stubs.c, and proto.h ("./ew_" by default).

(-s | --stubs)

Generate ew_stubs.c file containing stubbed-out versions of form and CGI functions normally provided by the application to facilitate rapid prototyping and development.

(-C | --no-compress)

Disable compression of the archive. (Overrides compress attribute in mime.types and _ACCESS files).

(-l | --little)

The byte-order of the generated archive data component. By default, a big-endian archive is generated. If --little is specified, then a little-endian

archive is generated.

(-P | --no-preprocessor)
Disable generation of `#line` pre-processor directives in generated `ew_code.c` file; normally, these directives are included so that the C compiler and debuggers can find the original C in the source HTML documents. This option should only be used if your C compiler does not understand the '`#line`' directive.

(-q | --quiet) | (-v | --verbose) | (-d | --debug)
The logging level. The --quiet options suppresses all but error messages. The `--verbose` is specified, more logging output is produced. The `--debug` is specified, still more logging output is produced. The default (non of these options specified) includes errors and some warning messages.

(-v | --version)
Displays the EmWeb/Compiler's version.

--
Two dashes in a row indicate the end of options. All remaining parameters are file and/or directory names. This is optional, as any parameter that does not begin with a '-' is assumed to be a file and/or directory name.

3.6. Compiler Output

The EmWeb/Compiler produces the following output files:

ew_data.dat
A raw binary file containing the data component of the archive.

ew_data.c
If the `--c` option is specified at the command line, this file is generated. It contains C code that, when compiled, defines an array of octets representing the data component of the archive and referenced by the global symbol `uint8 ew_archive_data[]`.

ew_code.c
A C file that, when compiled, defines the object component of the archive referenced by the global symbol `(EwsArchive) ew_archive`.

ew_proto.h
This C header file is generated if the archive contains EmWeb HTML forms or CGI documents. The header file defines function prototypes and data structures for an application-specific interface between EmWeb/Server and application-provided functions responsible for serving and submitting forms and/or raw CGI processing.

ew_stubs.c

If the `--stubs` option is specified at the command line, this file is generated. It contains stubbed-out versions of form and CGI functions normally provided by the application to facilitate rapid prototyping and development.

4. EmWeb/Server

The EmWeb/Server is written in portable ANSI C and is easily integrated into a wide variety of embedded application environments. EmWeb/Server implements the HTTP protocol and responds to requests from networked Web browsers with documents stored in run-time archives created by the EmWeb/Compiler.

The following figure illustrates a typical EmWeb/Server device management application.

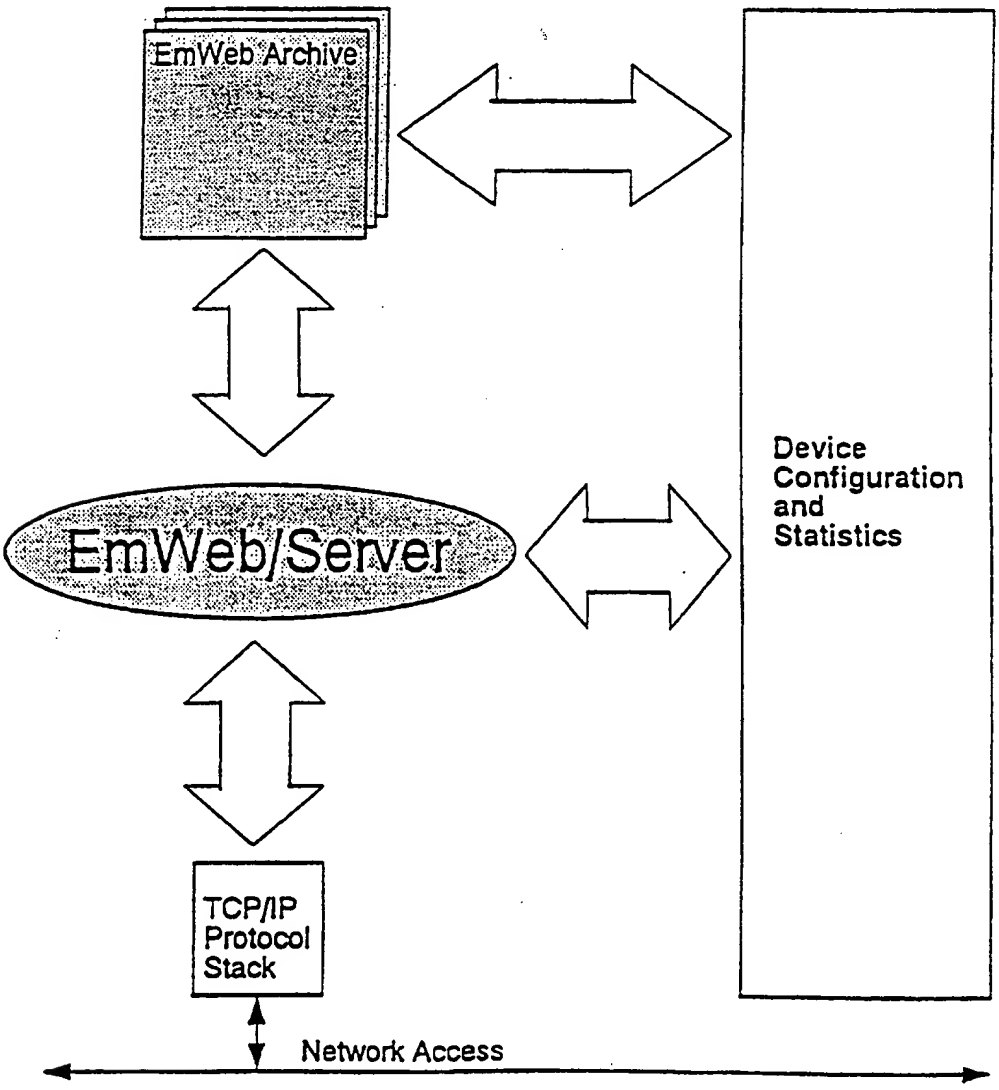


Figure 2: EmWeb/Server

4.1. Application Interfaces

The EmWeb/Server application interfaces are divided into six functional areas including system interfaces, network interfaces, document and archive maintenance, authorization and security, request context access, and raw CGI.

Many application interfaces are optional and can be configured out of the EmWeb/Server at compile time. This offers the system integrator flexibility in balancing the trade-offs between functionality, memory requirements, and system performance.

Most application interfaces are provided by the EmWeb/Server and are called by the application. However, some application interfaces must be provided by the application and are called by the EmWeb/Server. Function names beginning with "ews" and data types beginning with "Ews" are defined by the server for use by the application. Function names beginning with "ewa" and data types beginning with "Ewa" must be defined by the application for use by EmWeb/Server.

Status codes returned by the EmWeb/Server to the application are of type EwsStatus and are defined as follows (from `src/include/ews_def.h`):

```
/*
 * Status codes returned to the application by EmWeb/Server
 */
typedef enum EwsStatus_e
  {
    EWS_STATUS_OK,
    EWS_STATUS_BAD_MAGIC,
    EWS_STATUS_BAD_VERSION,
    EWS_STATUS_ALREADY_EXISTS,
    EWS_STATUS_NO_RESOURCES,
    EWS_STATUS_IN_USE,
    EWS_STATUS_NOT_REGISTERED,
    EWS_STATUS_NOT_CLONED,
    EWS_STATUS_NOT_FOUND,
    EWS_STATUS_AUTH_FAILED,
    EWS_STATUS_BAD_STATE,
    EWS_STATUS_BAD_REALM,
    EWS_STATUS_FATAL_ERROR,
    EWS_STATUS_ABORTED
  } EwsStatus;
```

Status codes returned by the application to the EmWeb/Server are of type EwaStatus and are defined as follows (from `src/include/ews_def.h`):

```
/*
 * Status codes returned to EmWeb/Server by the application
 */
typedef enum EwaStatus_e
  {
    EWA_STATUS_OK,
    EWA_STATUS_OK_YIELD,
    EWA_STATUS_ERROR
```



```
} EwaStatus;
```

4.1.1. System Interfaces

The system functional interfaces are further divided into four functional areas including initialization, scheduling, memory management, and time-of-day services.

4.1.1.1. Initialization and Shutdown

Before any other application interface function may be used, the EmWeb/Server must be initialized. This is accomplished by invoking the following function:

```
EwsStatus ewsInit ( void );
```

This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned. (EWS_STATUS_NO_RESOURCES is returned if EmWeb/Server was unable to allocate memory for internal hash tables, etc.).

A graceful shutdown of the EmWeb/Server can be accomplished by invoking the function below. This causes EmWeb/Server to terminate all outstanding HTTP requests and release all dynamically allocated resources.

```
EwsStatus ewsShutdown ( void );
```

This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned. (There are currently no conditions that cause an error to be returned).

4.1.1.2. Scheduling

The EmWeb/Server is capable of serving multiple HTTP requests simultaneously. It may be configured at compile-time to use either its own internal real-time scheduler or to make use of an external scheduler provided by the application's operating system.

If an external scheduler is to be used, HTTP requests may be multi-threaded if each TCP connection is handled by its own operating system thread. Otherwise, requests will be single-threaded. The `ewsRun()`, `ewsSuspend()` and `ewsResume()` functions described below for EmWeb/Server's internal scheduler would not be used. Instead, all application <EMWEB_STRING> and <EMWEB_INCLUDE> code fragments and `ewaFormServe_*` and `ewaFormSubmit_*` functions are invoked from the `ewsNetHTTPReceive()` function.

If the internal scheduler is to be used, HTTP requests are served from the following function:

```
EwsStatus ewsRun ( void );
```

This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned. (There are currently no conditions that cause an error to be returned). All application <EMWEB_STRING> and <EMWEB_INCLUDE> code fragments and

ewaFormServe_* and ewaFormSubmit_* functions are invoked from ewsRun().

The ewsRun() function processes any outstanding HTTP requests until either all requests have been served, or the application requests that EmWeb/Server yield the CPU as follows. Each time EmWeb/Server sends a buffer of HTTP response data to the network via the application-provided ewaNetHTTPSend function, the application may return the status EWA_STATUS_OK_YIELD to request that the EmWeb/Server relinquish control of the CPU. This will cause the EmWeb/Server to return from the ewsRun() function immediately. (Note that if the last buffer of the HTTP response was transmitted, the EmWeb/Server will close the connection and release any corresponding resources before returning control to the application).

In addition, the internal scheduler gives the application a means to temporarily suspend a specific HTTP request each time it invokes application-specific code from <EMWEB_STRING>, <EMWEB_INCLUDE>, and HTML form serve and submit functions. The application may instruct EmWeb/Server to suspend an HTTP request from the application-specific code provided in <EMWEB_STRING> and <EMWEB_INCLUDE> C code fragments, and EwaFormServe_* and EwaFormSubmit_* functions by invoking the following function before returning to the EmWeb/Server.

```
EwsStatus ewsSuspend ( EwsContext context );
```

This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned (EWS_STATUS_BAD_STATE). When the application returns to EmWeb/Server after invoking ewsSuspend, the EmWeb/Server may continue to process other outstanding requests from ewsRun() before returning control to the application. Note that the value returned by the application after calling ewsSuspend is ignored. The application's <EMWEB_STRING> or <EMWEB_INCLUDE> C code fragment or ewaFormServe_* or ewaFormSubmit_* function is re-invoked when the request is resumed. This functionality makes it possible to implement proxy servers (i.e. a local HTTP request might cause the application to send an IPC message to another process and suspend EmWeb/Server processing of this request until an IPC response is received). The application can resume a previously suspended request by invoking the following function.

```
EwsStatus ewsResume ( EwsContext context, EwaStatus status );
```

Status can be either EWA_STATUS_OK or EWA_STATUS_OK_YIELD. If the status is EWA_STATUS_OK, then the resumed request may be scheduled right away (for example, this could cause ewsRun to be invoked internally). Otherwise, the suspended request is scheduled to run, but is not actually run until control is transferred back to EmWeb/Server. This gives the application control over which CPU thread processes HTTP requests (i.e. a suspended request may be resumed by an interrupt handler for execution later in a polling loop that calls ewsRun). This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned (EWS_STATUS_BAD_STATE).

The following function may be invoked from within the application's <EMWEB_STRING> or <EMWEB_INCLUDE> C code fragment or ewaFormServe_* or ewaFormSubmit_* function to determine if this is the initial invocation of the application code, or if the code is being resumed.

```
boolean ewsContextIsResuming ( EwsContext context );
```

This function returns TRUE if the application code is being re-invoked after ewsResume. Otherwise, this function returns FALSE.

4.1.1.3. Memory Management

The EmWeb/Server requires the application to provide simple memory management for the allocation and release of temporary data-structures used throughout the processing of HTTP requests and maintaining the run-time document archive and authentication databases. These functions are equivalent to the POSIX malloc and free calls as follows:

```
void * ewaAlloc ( uintf bytes );
void ewaFree ( void * pointer );
```

The ewaAlloc function (or macro) returns a pointer to available memory of at least the specified number of bytes long, or NULL if no memory is available. The ewaFree function (or macro) returns a block of available memory previously allocated by ewaAlloc.

> **Note:** If the EmWeb/Server is unable to allocate memory during the processing of a particular HTTP request, the request is aborted, all resources related to the request are released, and the application's ewaNetHTTPEnd() function is invoked. Alternatively, the application may implement ewaAlloc() so as to block until resources are available using external operating system facilities.

4.1.1.4. Time-of-Day Management

Some of the optional HTTP/1.0 functionality requires a "time-of-day" for Date:, Expires:, and Last-modified: headers. Many embedded systems do not have a reliable time-of-day capability, so these features are configurable options in EmWeb/Server. To take advantage of these features, the application must provide the following function (or macro):

```
const char * ewaDate ( void );
```

This function should return the current time (GMT) in one of the following two formats:

RFC1123: (Preferred)

```
Fri, 08 Jun 1996 08:57:28 GMT
```


RFC850:

```
Friday, 08-Jun-96 08:57:28 GMT
```

4.1.2. Network Interfaces

The EmWeb/Server relies upon a pre-existing TCP/IP protocol stack available in the target software environment. The interface between the application's TCP/IP protocol stack and the EmWeb/Server is event driven. This approach makes it possible to port EmWeb/Server to a wide variety of software environments.

4.1.2.1. Network Buffer Management

The EmWeb/Server design assumes that the application maintains data buffers for the reception and transmission of TCP data.

Buffers can be uniquely identified by an application-defined buffer descriptor. No assumptions are made about the actual structure of the buffer descriptors or their relationship to the data. For example, a buffer descriptor could be an index into a table, a pointer to a structure (either contiguous or separate from the data represented), etc. The application is responsible for defining the appropriate type for `EwaNetBuffer` and a value for `EWA_NET_BUFFER_NULL`.

Buffers can be chained together. Given a buffer descriptor, EmWeb/Server must be able to get or set the "next buffer in the chain" field. This is done by the `ewaNetBufferNextSet` and `ewaNetBufferNextGet` functions (or macros). Note that the buffer chain is terminated when the next buffer value is `EWA_NET_BUFFER_NULL`.

```
EwaNetBuffer ewaNetBufferNextGet ( EwaNetBuffer buffer );

void ewaNetBufferNextSet ( EwaNetBuffer buffer, EwaNetBuffer next );
```

Given a buffer descriptor, EmWeb/Server can determine the start of data in the buffer. Additionally, EmWeb/Server may change the start of data in the buffer (EmWeb/Server only moves the pointer to the data upward (increments)). This is done by the `ewaNetBufferDataGet` and `ewaNetBufferDataSet` functions (or macros).

```
uint8 * ewaNetBufferDataGet ( EwaNetBuffer buffer );
void ewaNetBufferDataSet ( EwaNetBuffer buffer, uint8 *datap );
```

Given a buffer descriptor, EmWeb/Server can determine the size of contiguous data in the buffer. Additionally, EmWeb/Server may change the size of the buffer (EmWeb/Server only changes the size of the buffer downward (decrements)). This is done by the `ewaNetBufferLengthGet` and `ewaNetBufferLengthSet` functions (or macros).

```
uintf ewaN tBuff rLengthGet ( EwaNetBuffer buffer);
void ewaNetBufferLengthSet ( EwaN tBuffer buffer, uintf length );
```

EmWeb/Server may allocate a single buffer by invoking the `ewaNetBufferAlloc` function (or macro). The application may return a buffer of any size to EmWeb/Server. The size of the buffer must be initialized to the number of bytes available in the buffer for EmWeb/Server. If no buffers are available, this function returns `EWA_NET_BUFFER_NULL`. Additionally, EmWeb/Server may release a chain of one or more buffers by invoking the `ewaNetBufferFree` function (or macro).

```
EwaNetBuffer ewaNetBufferAlloc ( void );
void ewaNetBufferFree (EwaNetBuffer buffer);
```

Note: If the EmWeb/Server is unable to allocate a network buffer during the processing of a particular HTTP request, the request is aborted, all resources related to the request are released, and the application's `ewaNetHTTPEnd()` function is invoked. Alternatively, the application may implement `ewaNetBufferAlloc()` so as to block until resources are available using external operating system facilities.

4.1.2.2. TCP/IP Interfaces

The application is responsible for listening to the HTTP TCP port (80) for connection requests. When an HTTP TCP connection request is received, the application accepts the TCP connection on behalf of EmWeb/Server and invokes the following function:

```
EwsContext ewsNetHTTPStart ( EwaNetHandle handle );
```

This function returns a new context handle which the application is expected to use when referencing this HTTP request for future operations. The value `EWS_CONTEXT_NULL` is returned on failure (e.g. no resources available to handle the new request). The handle parameter is application-defined and is returned to the application unchanged by the EmWeb/Server in other network interface calls as illustrated below. The handle is also made available to `<EMWEB_STRING>` and `<EMWEB_INCLUDE>` C code fragments and `ewaFormServe_*` and `ewaFormSubmit_*` application functions by invoking the following function:

```
EwaNetHandle ewsContextNetHandle ( EwsContext context );
```

At any time, the application can abort an uncompleted HTTP request by invoking the following function:

```
EwsStatus  wsNetHTTPAbort ( EwsContext context );
```

This function is typically invoked by the application to notify EmWeb/Server that the corresponding TCP/IP connection has been disconnected. The function returns `EWS_STATUS_OK` on success. Otherwise, an error code is returned. (There are

currently no conditions that cause an error to be returned).

As data buffer(s) are received from the network on a TCP connection corresponding to an HTTP request, the application passes these buffers to the EmWeb/Server by invoking the following function.

```
EwsStatus ewsNetHTTPReceive (EwsContext context, EwaNetBuffer buffer);
```

This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned. (There are currently no conditions that cause an error to be returned). The buffer parameter describes a chain of one or more buffers containing raw TCP data. The structure of the buffers is determined by the application, and the application provides EmWeb/Server with functions to manipulate these buffers as defined previously. Note that EmWeb/Server takes responsibility for releasing received buffers when they are no longer needed, and may hold on to buffers containing certain HTTP request headers for the duration of the request.

When EmWeb/Server has assembled one or more buffers for transmission, it invokes the following function (or macro) which must be provided by the application:

```
EwaStatus ewaNetHTTPSend ( EwaNetHandle handle, EwaNetBuffer buffer );
```

This function transmits (and then releases) a chain of one or more network buffers and returns EWA_STATUS_OK or EWA_STATUS_OK_YIELD on success, or EWA_STATUS_ERROR on failure. If the application returns EWA_STATUS_OK, EmWeb/Server will continue processing outstanding requests in a round-robin fashion. If the application returns EWA_STATUS_ERROR, EmWeb/Server will abort the current request (as if ewsNetHTTPAbort() was called) and continue processing any other outstanding requests. The application returns EWA_STATUS_OK_YIELD to request that EmWeb/Server give up the CPU. In most cases, ewaNetHTTPSend() is called for each buffer's worth of HTTP response data generated by the EmWeb/Server. Thus, the CPU used by the server can be throttled by adjusting the buffer size available to the server and using the EWA_STATUS_OK_YIELD return code. (See 4.1.1.2. Scheduling, page 29). Note that use of this return code requires that the application eventually either reschedule processing of the request by invoking ewsRun, or abort the request by invoking ewsNetHTTPAbort.

The application may signal the EmWeb/Server with flow control events to indicate congestion on an outbound TCP connection. If a TCP connection's transmit window is full, the application may call the following function at any time:

```
EwsStatus ewsNetFlowControl ( EwsContext context );
```

This will cause the EmWeb/Server to temporarily suspend processing of the request. When the TCP connection's window opens again, the application may invoke the following function at any time to notify EmWeb/Server that processing of the request may continue:

```
EwsStatus ewsNetUnFlowControl ( EwsContext context);
```

If `ewsNetFlowControl()` is called from the `ewaNetHTTPSend()` function, then the application is responsible for transmitting the current buffer. However, `ewaNetHTTPSend()` will not be called again until the application has called `ewsNetUnFlowControl()`. Note that if `ewsNetFlowControl()` is invoked from outside `ewaNetHTTPSend()`, then EmWeb/Server may call `ewaNetHTTPSend()` once before suspending the request. In any case, the application should be prepared to receive notification of request completion as described below.

When the EmWeb/Server has completed processing an HTTP request, or if the application aborts an outstanding HTTP request by invoking `ewsNetHTTPAbort()`, it invokes the following function which must be provided by the application:

```
EwaStatus ewaNetHTTPEnd ( EwaNetHandle handle );
```

This function should close the TCP connection corresponding to the request and return `EWA_STATUS_OK` or `EWA_STATUS_OK_YIELD` on success or `EWA_STATUS_ERROR` on failure. Note that returning `EWA_STATUS_OK_YIELD` causes EmWeb/Server to give up the CPU. Otherwise, EmWeb/Server may continue processing other HTTP requests in progress.

If persistent connections are used, it is possible that several HTTP requests will be handled over a single TCP/IP connection. For some applications, it is necessary to reset application-specific state information and to release application-specific dynamic resources after each HTTP request. The following function may be provided by the application for this purpose:

```
void ewaNetHTTPCleanup ( EwaNetHandle handle );
```

This function must be provided by the application or defined as a empty macro. It is invoked by EmWeb/Server when a request completes, allowing the application to reset any processing state stored in the network handle. Note that for persistent connections, this routine may be called several times for the same connection, as one connection can be used for multiple requests. This routine will be called before `ewaNetHTTPEnd()` is invoked.

The EmWeb/Server must have access to an application-provided null-terminated character string representing the HTTP network location (e.g. "www.agranat.com:80". Note that the host name (left of the ':') may be a dotted decimal IP address. Also note that the ':<port-number>' may be omitted if the standard HTTP TCP port #80 is used. This string is returned by the following application-provided function (or macro).

```
const char * ewaNetLocalHostName ( EwsContext context );
```

4.1.3. Document and Archive Management

The EmWeb/Compiler generates an archive of one or more documents. Documents can be HTML files, JAVA programs, graphical images, or any other information resource addressable by a URL. Archives may be independently loaded or unloaded into the EmWeb/Server.

In most applications, the entire set of available documents are compiled into a single archive and loaded at boot time. However, some applications may desire to dynamically load archives at run-time as needed in order to reduce memory requirements. In fact, applications may implement a scheme similar to virtual memory page swapping under some operating systems to cache an active set of documents in memory while storing other documents in a secondary storage area. Such a secondary storage area could be in FLASH memory, disk, or on a remote server using TFTP or other protocols to load documents at run-time.

An EmWeb archive consists of two components. First, there is the archive data component containing the database of compressed documents, information about how to construct dynamic documents at run-time, access controls, etc. Second, there is the archive object component containing the run-time object code used for the construction of dynamic documents, etc.

Operating systems supporting the run-time loading and linking of object code may off-load both the data and object archive components to a secondary storage area. Otherwise, only the data components are off-loaded while the object components are statically linked into the run-time executable image.

Each archive contains an archive descriptor in the object component. The archive descriptor is referenced by a public symbol generated by the EmWeb/Compiler.

4.1.3.1. Installing and Removing Archives

In order to activate an archive at run-time, the application must invoke `ewsDocumentInstallArchive` with parameters indicating the address of the data component and the archive descriptor of the object component.

```
EwsStatus ewsDocumentInstallArchive
        ( EwsArchive descriptor, const uint8 *datap );
```

This function may return one of the following status codes:

`EWS_STATUS_OK`	The archive was installed successfully
`EWS_STATUS_BAD_MAGIC`	The data or object portion of the archive is invalid or corrupt. This may happen if the EmWeb/Server is configured for a big-endian archive when the EmWeb/Compil r generated a little-endian archive, or vice versa.
`EWS_STATUS_BAD_VERSION`	The archive generated by the EmWeb/Compiler is not understood by the EmWeb/Server.
`EWS_STATUS_ALREADY_EXISTS`	The archive has already been installed.

EWS_STATUS_IN_USE	The archive contains a document with a URL that has been previously loaded from a different archive..
EWS_STATUS_NO_RESOURCES	Insufficient dynamic memory is available to install the archive.

The archive may be deactivated by invoking ewsDocumentRemoveArchive.

```
EwsStatus ewsDocumentRemoveArchive ( EwsArchive descriptor );
```

These functions return EWS_STATUS_OK on success. Otherwise, EW_STATUS_IN_USE is returned indicating that the archive contains documents that are either being processed by outstanding HTTP requests or that have been cloned to other URLs.

The fixed data component of an archive contains the name of the archive and the date it was created by the EmWeb/Compiler. These values may be retrieved as follows:

```
const char * ewsDocumentArchiveName ( const uint8 *datap );
const char * ewsDocumentArchiveDate ( const uint8 *datap );
const char * ewsDocumentArchiveDate1036 ( const uint8 *datap );
```

4.1.3.2. Demand Loading

In order to implement on-demand archive loading, the application may register document URLs with the EmWeb/Server which are valid but not loaded. This is done by invoking the following function:

```
EwsDocument
ewsDocumentRegister (const char *url, EwaDocumentHandle handle);
```

This function returns an EmWeb/Server document descriptor of type EwsDocument, or EWS_DOCUMENT_NULL on error. The handle is an application-defined value passed back unchanged to the application in ewaDocumentFault as shown below.

If a registered document is requested by a Web browser, then EmWeb/Server notifies the application by invoking the following application-provided function:

```
EwaStatus
ewaDocumentFault ( EwsContext context, EwaDocumentHandle handle );
```

At this point, the application may load a new archive (possibly removing a previously installed archive to make room). When the archive containing the page is installed, the EmWeb/Server automatically completes processing the request when the archive is loaded. The request can be aborted either immediately by returning EWA_STATUS_ERROR from the ewaDocumentFault function, or by invoking ewsNetHTTPAbort.

Once a document is registered, there is no need to re-register it in the event that the

corresponding archive is removed. EmWeb/Server remembers that the document has been registered as dynamically loadable. However, the application may de-register a document by invoking the following function:

```
EwsStatus ewsDocumentDeregister ( EwsDocument document );
```

This function returns EWS_STATUS_OK on success. Otherwise, EWS_STATUS_IN_USE is returned indicating that the document was cloned or not registered.

4.1.3.3. Cloning

Documents may be cloned and assigned to a new URL. This allows multiple instances of a document to exist while minimizing the storage requirements. An application-specific handle can be used to identify an instance of a document from the request context.

The application clones a document by invoking the following function:

```
EwsDocumentewsDocumentClone
 ( const char *baseurl, const char *newurl, EwaDocumentHandle handle );
```

This function returns a document descriptor for the clone, or EWS_DOCUMENT_NULL on error.

The cloned document may be removed by invoking the following function:

```
EwsStatus ewsDocumentRemove ( EwsDocument document );
```

This function returns one of the following status codes:

EWS_STATUS_OK	The document was successfully removed
EWS_STATUS_NOT_CLONED	The document is not a clone.
EWS_STATUS_IN_USE	The document is itself cloned. The clone of the clone must be removed first.

All clones created from documents in an archive must be removed before the archive can be removed.

Note that the application's document handle is made available to the application in the HTTP request context by invoking the following function:

```
EwaDocumentHandle ewsContextDocumentHandle ( EwsContext context );
```

If the document corresponding to the context was not cloned or registered by the application, then this function returns EWA_DOCUMENT_HANDLE_NULL.

4.1.3.4. URL Rewriting

In some applications, it may be desirable to translate between URLs advertised to the outside world and URLs configured in the archive. The optional URL hook feature

provides the application with an opportunity to rewrite a requested URL before EmWeb/Server looks up the URL in its archive database. The following function (or macro) may be provided by the application for this purpose:

```
char * ewaURLHook ( EwsContext context, char * url );
```

This function passes the requested URL and request context to the application. The application may simply return url if no rewriting is desired, or it may return an absolute local URL to substitute. The returned URL is looked up in the archive and served as if requested directly. If the application returns NULL, the request is aborted.

This functionality is intended to allow the application to select a subdirectory of documents based upon requested language, content encodings, and other information available from the request context.

Note that this function may rewrite and return any portion of the url string parameter (provided that the string's length is not increased), or return a completely different string buffer instead. If a different buffer is used, the server copies the value immediately.

4.1.3.5. Document Data Access

In some applications, it may be desirable to access the content of a file pointed to by a URL directly (e.g. data files used by CGI scripts, etc.). The following function is optionally provided by the EmWeb/Server to the application for this purpose:

```
EwsStatus ewsDocumentData
        ( const char * url, uint32 *bytesp, const uint8 **datapp );
```

This function looks up the requested URL and, if found, updates `*bytesp` and `*datapp` with the length and pointer to the start of raw data in the file. (Note that the data is compressed if compression is enabled). Since EmWeb employs a proprietary compression technique, the application should specifically disable compression of files it wishes to access this way by using the `nocompress` attribute in the appropriate _ACCESS file (see 3.4. _ACCESS Files, page 22). This function returns EWS_STATUS_OK on success, or EWS_STATUS_NOT_FOUND on failure.

4.1.4. Authentication and Security

HTTP Authorization techniques are discussed in the HTTP specifications (see 7. References, page 87). Knowledge of these principles is essential for correct use of these authorization interfaces.

One or more HTML documents can be associated with a single "realm". A realm is a case-sensitive ASCII string that defines a "protection space" for the associated document(s).

Each realm may have zero or more authentication entries associated with it. If a realm has no authentication entries, then documents in that realm are not accessible.

A client that attempts to access a document associated with a non-null realm is required to authenticate itself. To authenticate, the client must send authentication parameters valid for at least one authentication entry for that document's realm. Clients that do not authenticate are denied access to the requested document.

For example, assume a realm exists called "foo". It has three authentication entries associated with it (two using the "basic cookie" scheme defined in the HTTP/1.0 specification, and one using the "digest" scheme defined in the HTTP/1.1 specification):

```
REALM: "foo"
        Authentication Entry 1:
             Type: "basic"
             parameters:   Username="user1"
                           Password="guest"
             EwaAuthHandle: <application's entry1 identifier>

        Authentication Entry 2:
             Type: "digest"
             parameters:   Username="user2"
                           Password="837I8U9"
                           DigestRequired=FALSE
             EwaAuthHandle:  <application's entry2 identifier>

        Authentication Entry 3:
             Type: "basic"
             parameters: Username="sinclair"
                         Password="babylon"
             EwaAuthHandle:  <application's entry3 identifier>
```

When a client attempts to access a document associated with realm "foo", it needs to authenticate against one of the above entries. Which one the client authenticates against is at its discretion.

When a client does authenticate against one of the above entries, the `EwaAuthHandle` for that entry is stored in the current context for the HTTP request (`EwsContext`). The datatype of this `EwaAuthHandle` is implementation-defined.

The following types are defined for the authorization application interface (from `src/include/ews_auth.h`):

```
/*
 * Defines the authorization schemes supported by the EmWeb server.
 * New schemes will be added as they are supported.
 */
typ def enum
{
  ewsAuthSch meBasic,
  ewsAuthSch meDigest,
  ewsAuthSchem ManualBasic,
  ewsAuthSch meManualDigest,
```



```
  ewsAuthMaxScheme            /* count of supported schemes */
} EwsAuthScheme;

/*
 * this structure defines parameters for all the supported
 * authorization types.  New parameters will be added in
 * to this structure in the future
 */
typedef union
{
#ifdef EW_CONFIG_OPTION_AUTH_BASIC
  struct
  {
        char  *userName;
        char  *passWord;
  } basic;
#endif /* EW_CONFIG_OPTION_AUTH_BASIC */
#ifdef EW_CONFIG_OPTION_AUTH_DIGEST
  struct
  {
        char    *userName;
        char    *passWord;

#       ifdef EW_CONFIG_OPTION_AUTH_DIGEST_M
       boolean digestRequired; /* require message data verification */
#       endif /* EW_CONFIG_OPTION_AUTH_DIGEST_M */
  } digest;
#endif /* EW_CONFIG_OPTION_AUTH_DIGEST */
  char reserved1;
} EwsAuthParameters;

/*
 * This structure represents a single authorization entry.
 */
typedef struct
{
  EwsAuthScheme     scheme;
  EwsAuthParameters params;
  EwaAuthHandle     handle;     /* user defined */
} EwsAuthorization;
```

An authorization entry may be created by invoking the following function:

```
EwsAuthHandle ewsAuthRegister
        ( const char *realm, const EwsAuthorization *authorization );
```

This function returns an EmWeb/Server authorization handle which corresponds to the authorization entry, or EWS_AUTH_HANDLE_NULL on failure. The entry contains the triplet (authorization scheme (e.g. basic cookie), scheme-specific parameters (e.g. user and password), application-specific authorization handle).

The application-specific authorization handle corresponding to the authentication validating a particular HTTP request is available from the context as follows:

```
EwaAuthHandle ewsContextAuthHandle ( EwsContext context );
```

The application may remove a previously registered authorization handle by invoking the following function:

```
EwsStatus ewsAuthRemove ( EwsAuthHandle authorization );
```

This function returns EWS_STATUS_OK on success. Otherwise, an error code is returned (there are presently no conditions that result in error).

By default, documents assigned to a non-null realm are protected and may only be accessed by authorized users. It may be desirable to enable or disable a realm's access controls. The following function disables access control for a realm making documents assigned to the realm accessible to everyone.

```
EwsStatus ewsAuthRealmDisable ( const char * realm );
```

This function returns EWS_STATUS_OK on success, or EWS_STATUS_BAD_REALM if the realm was not defined by any loaded archives.

The following function enables access control for a realm making documents assigned to the realm protected. Protected documents may only be accessed by authorized users.

```
EwsStatus ewsAuthRealmEnable ( const char * realm );
```

This function returns EWS_STATUS_OK on success, or EWS_STATUS_BAD_REALM if the realm was not defined by any loaded archives.

The following function (or macro) must be provided by the application. The function is invoked by EmWeb/Server to get a realm qualifier string from the application. The qualifier string is concatenated with the base realm name in authentication protocol messages such that browsers may differentiate between realms of the same name appearing in multiple servers.

```
const char *ewaAuthRealmQualifier ( void );
```

The application would typically return the string "@hostname", where hostname is the server's local host name or IP address. The application may return NULL if no qualifier is desired.

4.1.4.1. Basic Authentication

The Basic authentication scheme requires the browser client to transmit a Base64 encoded username and password to the server for authentication. While no less secure than SNMP v1, Basic authentication is vulnerable to network monitoring and replay attacks.

The authentication parameters for Basic authentication are simply the username and

password represented as null-terminated character strings.

4.1.4.2. Manual Basic Authentication

For some applications, it may be impossible to obtain the Basic authentication username/password pair prior to receiving requests for a document. Instead, the Basic authentication username/password may be stored in a remote secure database. This database would need to be queried for the username/password as each request requiring authentication is processed.

In order to support this, a special type of authentication scheme is defined: ewsAuthSchemeManualBasic. This scheme is available only if the build option `EW_CONFIG_OPTION_AUTH_MBASIC` is defined. Once defined, the application must provide the following interface:

```
boolean ewaAuthCheckBasic(EwsContext context
                          ,const char *realm
                          ,const char *basicCookie
                          ,const char *userName
                          ,const char *password );
```

The `ewsAuthSchemeManualBasic` scheme can be registered for any realm by using the `ewsAuthRegister` function. The `scheme` field of the `EwsAuthorization` parameter must be set to `ewsAuthSchemeManualBasic`, and the application handle can be set in the handle field. The params field is not used by the manual basic scheme. Manual basic can only be registered once per realm. If there are "non-manual" (`ewsAuthSchemeBasic`) Basic authorization entries registered to the same realm as a manual Basic entry, the "non-manual" authorization entries will be given preference. If no "non-manual" authorization entries match the request, then `ewaAuthCheckBasic` will be called.

The EmWeb/Server will invoke the application's `ewaAuthCheckBasic` function when authenticating a request for a realm protected by the manual basic scheme, unless a "non-manual" basic authorization entry matched. The application's `ewaAuthCheckBasic` function must now determine whether or not the authorization is valid. This function is passed the context, requested document's realm, and the base64-encoded basic cookie from the Authorization header (as specified in RFC 2068). If the build option `EW_CONFIG_OPTION_AUTH_MBASIC_DECODE` is defined, then `userName` and `password` will point to the username/password strings decoded from the `basicCookie`, otherwise these pointers are `NULL`. All strings are null terminated, and should not be modified by the application.

The return value from `ewaAuthCheckBasic` determines how the server will respond to the request. If `FALSE` is returned, access will be denied. If `TRUE` is returned, access will be granted .

Optionally, the application can suspend the context from within `ewaAuthCheckBasic` using `ewsSuspend`. EmWeb/Server will ignore the return value in this case. When the

context is later resumed by `ewsResume`, the `ewaAuthCheckBasic` function will be reinvoked by EmWeb/Server.

The application may abort the context from within `ewaAuthCheckBasic` by calling `ewsNetHTTPAbort`.

4.1.4.3. Digest Authentication

The Digest authentication scheme challenges the browser client using a "nonce" value. The client responds with a valid cryptographic checksum of the username, secret password, the given nonce value, the HTTP request method, and the requested URI. This avoids sending the secret password over the network. Furthermore, by generating unique nonce values, the server can make replay and other forms of attack impractical.

> **Note:** EmWeb's digest authentication support is derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.
>
> **Note:** Digest authentication is currently an internet proposed standard and is not yet supported by many browsers.

The authentication parameters for Digest authentication include the username and password represented as null-terminated character strings. In addition, one additional boolean parameter is defined as follows:

digestRequired
: If `TRUE`, the EmWeb/Server will require that the client provide a valid message digest to prevent forgery of the request message. This requires significantly more computation and is not supported by most browsers.

The application is required to define the application-specific structure called `EwaAuthNonce`. The `EwaAuthNonce` structure is application-specific and contains parameters to be used in calculating (via MD5 hash of the structure's contents) a unique nonce value. We recommend including the client's IP address, timestamp, a server-wide secret key, and any other random bits the application desires.

Two additional functions must be provided by the application for the generation and verification of nonce values as follows:

First, the `ewaAuthNonceCreate()` function is called by EmWeb/Server in response to an unauthenticated request for a document to request that the application initialize the `EwaAuthNonce` structure.

```
void
ewaAuthNonceCreate
        (EwsContext context
        ,const char *realm
        ,EwaAuthNonce *noncep );
```

The application should initialize the EwaAuthNonce structure. The request context and realm are provided as inputs. For example, a typical application could read the application-specific network handle from the request context to determine the client's IP address.

Second, the `ewaAuthNonceCheck()` function is called by EmWeb/Server in response to a client's request to authenticate against a nonce value previously initialized by the application in `ewaAuthNonceCreate()`.

```
typedef enum EwaAuthNonceStatus_e
  {
    ewaAuthNonceOK,      /* nonce value valid for request */
    ewaAuthNonceLastOK,  /* nonce value valid, but won't be again */
    ewaAuthNonceStale,   /* nonce value is stale, generate new nonce */
    ewaAuthNonceDenied   /* nonce value is invalid */
  } EwaAuthNonceStatus;

EwaAuthNonceStatus
ewaAuthNonceCheck
          (EwsContext context
          ,const char * realm
          ,EwaAuthNonce *noncep,uintf count );
```

This function is called by EmWeb/Server to verify that the nonce is valid for the current request. The `count` parameter indicates the number of times this nonce has been used previously (i.e. zero on the first call). The application may decide to accept the nonce by returning `ewaAuthNonceOK` or `ewaAuthNonceLastOK`. The `ewaAuthNonceLastOK` is used to signal to the server that the nonce value is valid this time, but will not be valid if used again. The server uses this information to generate a new nonce and pass a next nonce value to the browser to be used on the next request. Otherwise, the application may decide to expire the nonce (because it has been used too many times, because too much time has gone by since it was created, or any other reason that the application decides) by returning `ewaAuthNonceStale`, or refuse to authenticate (because the client IP address doesn't match) by returning `ewaAuthNonceDenied`.

4.1.4.4. Manual Digest Authentication

For some applications, it may be impossible to obtain the digest authentication username/password pairs prior to receiving requests for a document. Instead, the digest authentication username/password may be stored on a remote server. This server would need to be queried for the username/password as each request requiring authentication is processed.

In order to support this, a special type of authentication scheme is defined: `ewsAuthSchemeManualDigest`. This scheme is available only if the build option `EW_CONFIG_OPTION_AUTH_MDIGEST` is defined. Once defined, the application must provide the following interface:

```
bool an ewaAuthCheckDigest( EwsContext context
                          ,const char *realm
                          ,const EwaAuthNonce *noncep
                          ,const char *userName
                          ,const char **digest );
```

The ewsAuthSchemeManualDigest scheme can be registered for any realm by using the ewsAuthRegister function. The scheme field of the EwsAuthorization parameter must be set to ewsAuthSchemeManualDigest, and the application handle can be set in the handle field. The params.digest.digestRequired flag may be used if the EW_CONFIG_OPTION_AUTH_DIGEST_M build option is defined. All other parameters are not used by the manual digest scheme. Manual digest can be registered once per realm. If there are "non-manual" (ewsAuthSchemeDigest) digest authorization entries registered to the same realm as a manual digest entry, the "non-manual" authorization entries will be given preference. If no "non-manual" authorization entries match the request, then ewaAuthCheckDigest will be called.

The EmWeb/Server will invoke the application's ewaAuthCheckDigest function when authenticating a request for a realm protected by the manual digest scheme, unless a "non-manual" digest authorization entry matched. The application's ewaAuthCheckDigest function must now determine whether or not the authorization is valid. This function is passed the context, the requested document's realm, the nonce associated with this request, and the username as it appears in the request. All strings are null terminated, and should not be modified by the application.

ewaAuthCheckDigest returns a boolean status that determines how the server will respond to the request. If FALSE is returned, access will be denied. If TRUE is returned, the *digest parameter has been set to a null terminated string containing the ASCII string representation of the MD5 checksum of the username, realm, and password strings. This is referred to the H(A1) value in RFC 2069. For example, if the username is "user", password is "password", realm is "foo_realm", then *digest should be set to the string representation of MD5(user:foo_realm:password), which would be:

```
*digest = "eafd9339d0114895926c24ec79af7211"
```

The *digest value is used by the server to validate the response digest and entity-digest that are present in the request's Authorization header. If the validation does not pass, access will be denied. The memory used by the *digest value will remain referenced by the server until the context of the request is freed.

Optionally, the application can suspend the context from within ewaAuthCheckDigest using ewsSuspend. The return value will be ignored. When the context is later resumed with ewsResume, the ewaAuthCheckDigest function will be reinvoked.

The application may abort the context from within ewaAuthCheckDigest by calling ewsNetHTTPAbort.

4.1.4.5. Application Security Verification

Some applications may provide their own proprietary form of client authentication. These proprietary methods will vary from application to application. For example, an application could use the client's IP address to restrict the client's access to certain realms.

EmWeb provides a generic interface that allows an application to provide any proprietary checks on the request context and requested realm. This interface allows the application to deny service to the client if so desired.

This interface is made available by defining the `EW_CONFIG_OPTION_AUTH_VERIFY` build option. The application must then provide the following interface:

```
boolean ewaAuthVerifySecurity( EwsContext context
                              ,const char *realm );
```

This function is called by EmWeb/Server after the current request has passed the standard authorization procedures described above. For example, if basic authentication is enabled on a realm, `ewaAuthVerifySecurity` will be called only after the basic authentication *successfully* completes. If digest authentication is used, then `ewaAuthVerifySecurity` is called after the response digest is *successfully* verified, but before the entity-digest is checked (if the `EW_CONFIG_OPTION_AUTH_DIGEST_M` build option is turned on).

`ewaAuthVerifySecurity` is passed the current context, and a pointer to a null terminated string containing the realm. The application can allow the request to complete by returning `TRUE` from `ewaAuthVerifySecurity`. If `ewaAuthVerifySecurity` returns `FALSE`, the request is denied access.

`ewaAuthVerifySecurity` is provided only as a secondary security mechanism, and can only be used in conjunction with a defined standard security scheme, such as basic or digest.

Optionally, the application can suspend the context from within `ewaAuthVerifySecurity` using `ewsSuspend`. The return value will be ignored. When the context is later resumed, the `ewaAuthVerifySecurity` function will be reinvoked.

The application may abort the context from within `ewaAuthVerifySecurity` by calling ewsNetHTTPAbort.

4.1.4.6. Document Realm Assignment

The authentication realm of a document is typically determined from the `_ACCESS` configuration files by the EmWeb/Compiler. However, it may be desirable to change the document's realm assignment at run-time. The following function is optionally provided by the EmWeb/Server to the application for this purpose:

```
EwsStatus ewsDocumentSetRealm ( const char *url, const char *realm );
```

This function returns EWS_STATUS_OK on success, or EWS_STATUS_NOT_FOUND if the specified local URL is not installed, or EWS_STATUS_NO_RESOURCES if insufficient resources were available to create a new realm.

4.1.5. Request Context Access

Each HTTP request received by the EmWeb/Server is assigned a unique context. A context handle of type EwsContext is first returned to the application from the ewsNetHTTPStart function.

Application code responsible for the run-time content of HTML documents (i.e. C code fragments from <EMWEB_STRING> and <EMWEB_INCLUDE>, HTML form processing functions ewaFormServe_* and ewaFormSubmit_*, and raw CGI interface functions ewaCGIStart_* and ewaCGIData_*) have access to the corresponding HTTP request context. In the case of <EMWEB_STRING> and <EMWEB_INCLUDE> C code fragments, the context is available from the local variable symbol ewsContext. In the case of the form and raw CGI processing functions, the context is passed as a parameter.

Several context access functions are defined which can be called by the application to extract information about the current context. For example, the function:

```
EwaNetHandle ewsContextNetHandle ( EwsContext context )
```

Returns the network handle corresponding to the HTTP context that was passed by the application to the EmWeb/Server in the ewsNetHTTPStart function. The following table lists these context access functions:

Context Access Function	Description
`ewsContextNetHandle` `ewsContextNetHandle ( EwsContext context );`	Returns the network handle that was passed by the application to EmWeb/Server in the ewsNetHTTPStart function.

`EwaDocumentHandle`
`ewsContextDocumentHandle`
`( EwsContext context );`
	Returns the document handle that was passed by the application to EmWeb/Server in either the ewsDocumentRegister or ewsDocumentClone functions, or EWA_DOCUMENT_HANDLE_NULL.

`EwaAuthHandle`
`ewsContextAuthHandle`
`( EwsContext context );`
	Returns the authorization handle that was passed by the application to EmWeb/Server in the ewsAuthRegister function, or EWA_AUTH_HANDLE_NULL.

Context Access Function	Description
`boolean ewsContextIsResuming ( EwsContext context );`	Returns `TRUE` if this is the first application call-out after a suspended request was resumed by calling `ewsResume`. Otherwise, returns `FALSE`.
`uint32 ewsContextIterations ( EwsContext context );`	Returns the iteration count (starting with zero) to an `<EMWEB_STRING>` or `<EMWEB_INCLUDE>` C code fragment with the `EMWEB_ITERATE` attribute specified.
`uintf ewsContextDate ( EwsContext context, char *datap, uintf length );`	Copies the HTTP `Date:` header into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsContextPragma ( EwsContext context, char *datap, uintf length );`	Copies the HTTP `Pragma:` header into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsContextFrom ( EwsContext context, char *datap, uintf length );`	Copies the HTTP `From:` header into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsContextIfModifiedSince ( EwsContext context, char *datap, uintf length );`	Copies the HTTP `If-modified-since:` header into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsContextReferer ( EwsContext context, char *datap, uintf length );`	Copies the HTTP `Referer:` header into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsContextUserAgent ( EwsContext context, char *datap, uintf length );`	Copies the HTTP `User-agent:` value into the buffer provided by the application and returns the actual number of bytes present.

Context Access Function	Description
`uintf ewsContextHost ( EwsContext context, char *datap, uintf length );`	Copies the HTTP Host: value into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsCGIServerProtocol( EwsContext context, char *datap, uintf length );`	Copies the HTTP server protocol "`HTTP/1.0`" or "`HTTP/0.9`" into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsCGIRequestMethod( EwsContext context, char *datap, uintf length );`	Copies the HTTP request method "`GET`", "`HEAD`", or "`POST`" into the buffer provided by the application and returns the actual number of bytes present.
`const char * ewsCGIPathInfo( EwsContext context );`	Returns pointer to CGI extra path information if present, or NULL.
`const char * ewsCGIScriptName( EwsContext context );`	Returns pointer to URL base path string.
`uintf ewsCGIQueryString( EwsContext context, char *datap, uintf length );`	Copies the URL query string into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsCGIContentType( EwsContext context, char *datap, uintf length );`	Copies the HTTP `Content-type:` value into the buffer provided by the application and returns the actual number of bytes present.
`uintf ewsCGIContentLength( EwsContext context );`	Returns the value of the HTTP `Content-length:` header or zero if not present.
`uintf ewsCGIContentEncoding( EwsContext context, char *datap, uintf length );`	Copies the HTTP `Content-encoding:` value into the buffer provided by the application and returns the actual number of bytes present.

Note that the context access functions that copy values into application-provided strings return the actual number of bytes in the value. If the value is not present, then these functions return zero. These functions do not overwrite the application buffer (i.e. the `length` parameter shall be honored by EmWeb/Server). However, the function

may return a number greater than the `length` if the application buffer was not large enough to accommodate the value. In fact, the `length` parameter of zero may be used to determine the size of the buffer needed.

4.1.6. Raw CGI

We believe that one of the greatest features of EmWeb/Server is that the system integrator can implement nearly any embedded Web-based application without using CGI.

While Agranat Systems strongly discourages its use, we recognize that some highly sophisticated applications may require a raw CGI interface, and so we provide one.

A URL corresponding to the base name of a CGI script may be specified in the `_ACCESS` file which is processed by the EmWeb/Compiler. The application must provide two functions for each CGI script with names derived from the value of the `cgi-symbol` directive in the `_ACCESS` file as follows:

```
EwaCGIHandle ewaCGIStart_symbol ( EwsContext context );
```

This function is called by EmWeb/Server when a CGI script is first requested by a Web server. The application returns a handle that is passed back to the application in the second function:

```
void ewaCGIData_symbol ( EwaCGIHandle handle, EwaNetBuffer buffer );
```

This second function is called by EmWeb/Server to pass one or more buffers to the application containing raw CGI data (i.e. the HTTP message body) as they are received from the network. The application is responsible for knowing where the end of data is from the `Content-length:` header (available from `ewsCGIContentLength`).

> **Restriction:** The application's `ewaCGIData_symbol` function must not attempt to access request header information from the request context because the network buffer(s) containing request headers may have been discarded. Any information needed by the application should be copied and saved during `ewaCGIStart_symbol`.

The application may generate a standardized HTTP response header by invoking the following function:

```
EwsStatus ewsCGISendStatus
 ( EwsContext context, const char * status, const char *string);
```

This function sends an HTTP status line for the appropriate HTTP version. The status argument must contain a null-terminated character string containing a three-digit HTTP status code followed optionally by descriptive status text. If not NULL, the string argument contains a null-terminated character string containing additional headers, and/or data. This function returns `EWS_STATUS_OK` on success. Otherwise, an error

code (EWS_STATUS_BAD_STATE or EWS_STATUS_NO_RESOURCES) is returned. Note that if ewsCGISendStatus is to be used, it must be used only once per request, and must be used before calling ewsCGIData.

For example, to send a simple HTML page:

```
ewsCGISendStatus ( context, "200 OK",
        "Content-Type: text/html\r\n"
        "Content-Length: 66\r\n"
        "\r\n"
        "<HEAD><TITLE>Example</TITLE></HEAD>"
        "<BODY>This is an example</BODY>" );
```

The application may send additional headers followed by a message body by invoking the following function:

```
EwsStatus ewsCGIData ( EwsContext context, EwaNetBuffer buffer );
```

This function sends the chain of buffers to the Web browser. Note that the application must delimit the header and body sections with a blank line sequence in accordance with the HTTP specification. To complete the request, the application should invoke this function with an EWA_NET_BUFFER_NULL buffer descriptor. This causes EmWeb/Server to close the connection. This function returns EWS_STATUS_OK on success. Otherwise, an error code (EWS_STATUS_BAD_STATE) is returned.

Instead of sending headers and data, the application may instruct EmWeb/Server to generate a redirect to a specified URL or, if local, serve a local document as a response. To send a redirect response to the browser (for a local or remote document) the application may invoke the following function:

```
EwsStatus ewsCGIRedirect ( EwsContext context, const char * url );
```

To serve a local document as a response, the application may invoke the following function:

```
EwsStatus ewsContextSendReply ( EwsContext context, char * url );
```

Note that these functions may not be used in conjunction with ewsCGIData or ewsCGISendStatus. The url parameter may be a relative or absolute URL.

Several CGI environment values are available from the HTTP request context as outlined in the Context section above. In addition, the following two functions return global server information for conformance with CGI/1.0:

```
const char * ewsCGIServerSoftware ( void )
```

Returns the server software version string, for example "Agranat-EmWeb/R1_0".

```
const char * ewsCGIGatewayInterface ( void )
```

Returns the gateway interface version string, for example "CGI/1.0".

4.1.7. Logging Hook

The application may optionally provide the following function (or macro) for logging HTTP events:

```
void ewaLogHook(EwsContext context, EwsLogStatus status);
```

This function is invoked by EmWeb/Server at least once for each HTTP request, and possibly a second time for certain dispositions after a successful request. Possible status values include (from src/include/ews_def.h):

```
typedef enum EwsLogStatus_e
 {
   /*
    * 200 Request accepted
    */
   EWS_LOG_STATUS_OK,

   /*
    * Request dispositions (after successful request)
    */
   EWS_LOG_STATUS_NO_CONTENT,                  /* 204 no-op form or ima
    EWS_LOG_STATUS_MOVED_PERMANENTLY,            /* 301 link */
    EWS_LOG_STATUS_MOVED_TEMPORARILY,            /* 302 redirect */
    EWS_LOG_STATUS_SEE_OTHER,                    /* 303 see other */
   EWS_LOG_STATUS_NOT_MODIFIED,                /* 304 not modified sinc

   /*
    * 401 Unauthorized
    */
   EWS_LOG_STATUS_AUTH_FAILED,                 /* authorization failed
   EWS_LOG_STATUS_AUTH_FORGERY,                /* bad message checksum
   EWS_LOG_STATUS_AUTH_STALE,                  /* authorization nonce s
   EWS_LOG_STATUS_AUTH_REQUIRED,               /* authorization not pre
   EWS_LOG_STATUS_AUTH_DIGEST_REQUIRED,        /* message digest not pr

   /*
    * 400 Bad Request
    */
   EWS_LOG_STATUS_BAD_REQUEST,                   /* HTTP parse error */
   EWS_LOG_STATUS_BAD_FORM,                    /* form data parse error
   EWS_LOG_STATUS_BAD_IMAGEMAP,                /* imagemap query parse

   /*
    * Additional errors
    */
   EWS_LOG_STATUS_NOT_FOUND,                   /* 404 not found or hidd
   EWS_LOG_STATUS_METHOD_NOT_ALLOWED,          /* 405 method not allowe
   EWS_LOG_STATUS_LENGTH_REQUIRED,             /* 411 length required *
   EWS_LOG_STATUS_UNAVAILABLE,                 /* 503 aborted Document
   EWS_LOG_STATUS_NOT_IMPLEMENTED,             /* 501 bad method for UR
```

```
        EWS_LOG_STATUS_NO_RESOURCES,          /* 500 insufficient reso
        EWS_LOG_STATUS_INTERNAL_ERROR         /* 500 internal error */

    } EwsLogStatus;
```

4.1.8. Local Filesystem Interfaces

The EmWeb Server provides the ability to manage a filesystem on the serving system. The Server supports:

- uploading a file from a client (browser) to the server

- serving the contents of a file in response to a URL GET

In both cases, the filesystem is external to the EmWeb Server. EmWeb assumes nothing about the implementation of this filesystem; it can exist on a local disk, non-volatile memory, or on an external proxy. All EmWeb requires is the ability to:

- retrieve/set meta-information about the file (e.g., type, size, modification time, etc.).

- open a file

- read a number of bytes from a existing file

- write a number of bytes to a newly created file

- close a file.

The API that must be supplied to EmWeb to provide these functions is defined below. To enable the basic file support, define the preprocessor symbol `EW_CONFIG_OPTION_FILE` in `ew_config.h`.

4.1.8.1. The EwsFileParams structure

The `EwsFileParams` structure is defined in the `ews_sys.h` include file. It is used by the EmWeb server to associate meta-information with a local file. The `EwsFileParams` structure consists of a union of two substructures. These substructures correspond to the file function that EmWeb is to perform. The `fileField` structure is used when the browser is uploading a file to the server via a form `POST` (see RFC1867). The `fileInfo` structure is used to serve a file in response to a `GET` for a particular URL. The individual fields for each of these structures is described in the description associated file operation.

```
typedef union EwsFileParams_s
{
  /*
   * fileField - for support of the form INPUT TYPE=FILE field.
   * The server fills out this structure, and passes it to
   * the application when the file is submitted
```

```
   */
# ifdef EW_CONFIG_OPTION_FIELDTYPE_FILE
  struct
  {
    const char *fileName;         /* file name or NULL */
    const char *contentType;      /* MIME type */
    const char *contentEncoding;  /* content encoding or NULL */
    const char *contentDisposition; /* Content-disposition: */
    int32       contentLength;    /* length or EWS_CONTENT_LENGTH_UNKNOWN */
  } fileField;
# endif /* EW_CONFIG_OPTION_FIELDTYPE_FILE */


  /*
   * fileInfo - for support for local file operations (GET,HEAD,OPTIONS,
   * PUT,DELETE). This structure is setup by the application when a URL
   * that corresponds to a local file is received.  This structure
   * gives the server all it needs to know to handle the file operation.
   */
# if defined( EW_CONFIG_OPTION_FILE_GET )
  struct
  {
    EwaFileName fileName;         /* file name (opaque) */
    const char *contentType;      /* MIME type */
    const char *contentEncoding;  /* content encoding or NULL */
    const char *contentLanguage;  /* content language or NULL */
    const char *eTag;             /* HTTP/1.1 cachability tag, or NULL */
    const char *lastModified;     /* modification time (RFC1123) or NULL */
    const char *lastModified1036;/* modification time (RFC1036) or NULL */
    const char *realm;            /* auth realm or NULL */
    int32       contentLength;    /* length or EWS_CONTENT_LENGTH_UNKNOWN */
    EwsRequestMethod   allow;     /* allowed methods */
    /* HEAD & OPTION _always_ allowed by default */

  } fileInfo;

  char reserved1;

} EwsFileParams, *EwsFileParamsP;
```

4.1.8.2. File Upload

Support for File Upload is enabled in the EmWeb server by defining the preprocessor symbol `EW_CONFIG_OPTION_FIELDTYPE_FILE` in `ew_config.h`.

To support file upload, EmWeb supports the methods described in RFC1867 (Form-based File Upload in HTML). At the time of this writing, only Netscape Navigator implements this RFC. For more information about RFC1867 and file upload support, contact Agranat Systems.

RFC1867 describes a new form input field of `TYPE=FILE`. This form field results in an input field that allows a user to enter a filename. This filename is the name of a file that is accessible by the browser. On submission of this form, the browser reads the

file, and sends the file's contents to server as part of the POST request.

This file field takes the following form:

```
<INPUT TYPE=FILE NAME=name VALUE=value>
```

Where *name* is the name of the form field and value is the default filename to display in the form when it is first served. When the EmWeb/Compiler encounters one of these form fields, the following form structure fields are generated:

```
char            *name;
EwaFileHandle   name_handle;
```

The *name* field is used by the form serve function only. It allows the application to override the default file name given when the form is first displayed (note: some browsers do not support a default value - call Agranat Systems for more information). The *name*_handle field is used by the form submit function only. It provides the file handle for the submitted file (see the discussion of ewaFilePost below). The status for name_handle is set to (EW_FORM_RETURNED | EW_FORM_DYNAMIC) on successful file POST, else EW_FORM_FILE_ERROR.

When EmWeb receives the submitted file, it allocates an EwsFileParamS structure and initializes the fileField substructure. The fileField substructure is setup using the optional file headers that are sent along with the file. The fileField fields are:

fileName
: Null terminated string containing the name of the file. This is usually supplied as the basename of the file (no path information). This field will be a NULL pointer if the browser does not supply a filename.

contentType
: Null terminated string containing the value of the Content-Type header, or a NULL pointer if the Content-Type header is not present (assumed to be text/plain).

contentEncoding
: Null terminated string containing the value of the Content-Encoding header, or a NULL pointer if the Content-Encoding header is not present.

contentDisposition
: Null terminated string containing the value of the Content-Disposition header, or a NULL pointer if the Content-Disposition header is not present.

contentLength
: set to the number of bytes in the file to be posted. If the Content-Length header is not supplied by the browser, then this field will be set to

EWS_CONTENT_LENGTH_UNKNOWN.

As soon as EmWeb receives the submitted form containing an INPUT TYPE=FILE field with file content, prior to calling the application's submit function, a call will be made to the application interface ewaFilePost:

```
EwaFileHandle ewaFilePost(
        EwsContext context,
        const EwsFileParams *params );
```

This function is provided by the application. The application defines the appropriate type for EwaFileHandle, and the value for EWA_FILE_HANDLE_NULL, which indicates a NULL EwaFileHandle. The params parameter is initialized by EmWeb as described above, and released by EmWeb on return from ewaFilePost.

ewaFilePost should open a file, and return a EwaFileHandle value that uniquely identifies that file. On error, ewaFilePost can return EWA_FILE_HANDLE_NULL, which will cause EmWeb to discard the contents of the submitted file without discarding the other submitted form fields. The application can also call ewsSuspend to suspend the current context. In this case, ewaFilePost will be reinvoked (with ewsContextIsResuming(context) == TRUE) once the application calls ewsResume on the context. The request can be aborted and the connection closed if ewaFilePost invokes ewsNetHTTPAbort.

After ewaFilePost is called, EmWeb will start calling the file write routine:

```
sintf ewaFileWrite( EwsContext context,
                    EwaFileHandle handle,
                    const uint8 *datap,
                    uintf length );
```

handle is set to the value that was returned by the ewaFilePost call. datap is a pointer to the data to be written, and length is set to the number of bytes that can be written from datap. ewaFileWrite returns the number of bytes written, or < 0 if an error occurred. If < 0 is returned, EmWeb will discard the rest of the incoming file without discarding the other submitted form fields. The application can suspend the current context by calling ewsSuspend from within ewaFileWrite and returning the number of bytes written prior to suspending (can be zero). The application can then resume the context by calling ewsResume. This will cause ewaFileWrite to be reinvoked with ewsContextIsResuming(context) == TRUE and datap and length adjusted by the number returned by the suspended call to ewsFileWrite. The application can also call ewsNetHTTPAbort to abort the entire request, and close the connection.

After the entire file has been received and written, the application's form submit function will be called. If the status field has been set to EW_FORM_RETURNED, then the *name*_handle contains the file handle for the file as returned by ewaFilePost. It is the

responsibility of the form submit function to close the file at this point.

Note well that once a file handle is returned by `ewaFilePost`, it is the responsibility of the application to close that file handle during the form submit function. EmWeb/Server will close the file handle if the request is aborted prior to calling the submit

4.1.8.3. File Serve

The EmWeb Server allows an application to provide a file external to an EmWeb archive for the response to a GET request. This functionality is enabled by defining the preprocessor symbol `EW_CONFIG_OPTION_FILE_GET` in `ew_config.h`.

To serve a file, an application must "mark" a GET request URL as representing a non-archive file. Once this is done, the EmWeb Server will "open" this file, read data from it and serve this data in response to the URL GET. After all data is served, EmWeb Server will close the file.

To "mark" a URL as a local file, the application calls the server interface `ewsContextSetFile` during a URL GET:

```
EwsStatus ewsContextSetFile( EwsContext context,
                             EwsFileParamsP params );
```

`ewsContextSetFile` can be called from:

- `ewaURLHook` once the URL is determined to be a local file entity.
- an application's form submit function prior to calling `ewsContextSendReply`. In this case, the URL passed to `ewsContextSendReply` represents the local file to be served.

Note that both the `ewaURLHook` and an application's form submit function can suspend the current context using `ewsSuspend`. This allows the application to do any asynchronous tasks needed prior to calling `ewsContextSetFile`.

The `fileInfo` substructure of the `param` parameter must be set up by the application prior to calling `ewsContextSetFile`. This structure gives EmWeb Server all the information about the file it needs in order to serve the file as response content. The fields are described below:

EwaFileName fileName
: This field is for use only by the application, EmWeb Server ignores this field. The type `EwaFileName` must be specified by the application.

contentType
: set to a NULL terminated string containing the value for the `Content-Type` header that is to be sent with the file.

contentEncoding
: set to a NULL terminated string containing the value for the Content-Encoding header that is sent with the file. If a NULL pointer, no Content-Encoding header is sent.

contentLanguage
: set to a NULL terminated string containing the value for the Content-Language header that is sent with the file. If a NULL pointer, no Content-Language header is sent.

eTag
: reserved, must be set to NULL.

lastModified
: the last modification date of the file in RFC1123 format as a NULL terminated string. If NULL, it is assumed that the file should be served each time it is requested.

lastModified1036
: the last modification date of the file in RFC1036 format as a NULL terminated string. Some browsers cannot correctly recognize the RFC1123 format, so this field should be set in addition to the lastModified field. Call Agranat Systems for more information.

realm
: the realm the request should be authenticated against for this file as a NULL terminated string, else NULL if no authorization required. Authorization is only checked when the file is served as a result of calling ewsContextSetFile from ewaURLHook - the realm used to protect the form submission is used implicitly when serving the file as a result of a ewsContextSendReply.

contentLength
: Set to the number of bytes in the file, else EWS_CONTENT_LENGTH_UNKNOWN if the file's length is unknown.

allow
: reserved, must be set to ewsRequestMethodGet.

EmWeb/Server references the param parameter for the duration of the file operation. As such, this parameter must be accessible throughout the life of the request - beyond the context of the ewaContextSetFile call. It can be safely deallocated by the application during the ewaNetHTTPCleanup call.

Once the URL has been "marked" by the ewsContextSetFile call, EmWeb/Server will open the file using the ewaFileGet interface:


```
EwaFileHandle ewaFil Get( EwsContext context,
                          const char *url,
                          const EwsFileParamsP params );
```

The `url` parameter is the URL that was "marked" as a file access. `params` is the `EwsFileParams` structure that was passed to `ewsContextSetFile`. The application defines the appropriate type for `EwaFileHandle`, and the value for `EWA_FILE_HANDLE_NULL`, which indicates a `NULL EwaFileHandle`.

`ewaFileGet` should open the file identified by `params` and `url`, and return a `EwaFileHandle` that uniquely represents the file. On error, `ewaFileGet` can return `EWA_FILE_HANDLE_NULL`. This causes the connection to abort. If desired, the application can suspend the current context by calling `ewsSuspend` from within `ewaFileGet`. The context can be resumed at a later time by calling `ewsResume`. `ewaFileGet` will be reinvoked (with `ewsContextIsResuming( context ) == TRUE`).

Once the file has been opened by `ewaFileGet`, EmWeb/Server will read the file by calling `ewaFileRead`:

```
sintf ewaFileRead( EwsContext context,
                   EwaFileHandle handle,
                   uint8 *datap,
                   uintf length );
```

This function should copy up to `length` bytes into the area of memory starting at `datap`. `handle` is set to the file handle returned by `ewaFileGet`. On return, `ewaFileRead` should return the number of bytes written into `datap`, which can be less than `length`. On error, `< 0` should be returned (this causes the request to be aborted and the connection close). This function can suspend by calling `ewsSuspend` and returning the number of bytes written into `datap` (can be zero). When the file described by `handle` has been completely read, this routine must return zero. Returning zero indicates that End-of-File has been reached (except when `ewsSuspend` has been called prior to returning).

Once all file data has been read, EmWeb/Server will close the file by calling ewaFileClose:

```
void ewaFileClose ( EwaFileHandle handle, EwsStatus status );
```

`ewaFileClose` should close the file associated with `handle` (the `EwsFileParams` structure can be safely deallocated at this point). `status` will be set to `EWS_STATUS_OK` on successful completion of the file serve, otherwise it will be set to an error code. EmWeb/Server will call this function to close a file handle during an aborted request prior to calling `ewaNetHTTPEnd`. Note that this function does not support suspension.

4.2. Application Interface Examples

This section is intended to illustrate the use of application interface functions under various scenarios.

The following tables illustrates the application interfaces used in the simplest EmWeb/Server configuration.

Application->Server (Initialization)	Server -> Application	Description
`ewsInit`	`ewaAlloc` `...`	Application initializes EmWeb/Server. The server may allocate run-time memory resources.

`ewsDocumentInstallArchive`
`...`
`ewsAuthRegister`
`...`
	`ewaAlloc`
	`...`
		Application installs one or more document archives and registers zero or more authenticated users. The server allocates run-time memory resources for its internal databases.

Application->Server (HTTP Transaction)	Server -> Application	Description
`ewsNetHTTPStart`	`ewaAlloc` `...`	Application accepts a new HTTP TCP connection request and informs the EmWeb/Server. The server allocates memory resources for the request.

`ewsNetHTTPReceive`
`...`
	`ewaAlloc`
	`...`
	`ewaNetBufferFree`
	`...`
		Application accepts request data from the network and passes it to the server for processing. The server may allocate memory resources to process the request as needed. Some network buffers received may be released at this point.

Application->Server (HTTP Transaction)	Server -> Application	Description
ewsRun	<EMWEB_STRING> <EMWEB_INCLUDE> ewaFormServe_* ewaFormSubmit_* ewaCGIStart_* ewaCGIData_* ...	The server invokes the application code attached to the requested document in order to construct a response on-the-fly.
	ewaNetBufferAlloc ... ewaNetHTTPSend (STATUS_OK) ...	The server allocates network buffers as needed, fills them with HTTP response data, and sends them to the application for transmission.
	ewaFree ... ewaNetBufferFree ... ewaNetHTTPEnd	The server releases any remaining resources associated with the HTTP transaction and terminates the request. The application closes the TCP connection at this point.

The next table illustrates a typical HTTP transaction during which the application instructs the EmWeb/Server to yield control of the CPU.

Application->Server (Scheduling)	Server -> Application	Description
ewsNetHTTPStart	ewaAlloc ...	Application accepts a new HTTP TCP connection request and informs the EmWeb/Server. The server allocates memory resources for the request.
ewsNetHTTPReceive ...	ewaAlloc ... ewaNetBufferFree ...	Application accepts request data from the network and passes it to the server for processing. The server may allocate memory resources to process the request as needed. Some network buffers received may be released at this point.
ewsRun	<EMWEB_STRING> <EMWEB_INCLUDE> ewaFormServe_* waFormSubmit_* ...	The server invokes the application code attached to the requested document.

Application->Server (Scheduling)	Server -> Application	Description
ewsSuspend		The application suspends the request. Processing on the request stops until resumed.
ewsResume	<EMWEB_STRING> <EMWEB_INCLUDE> ewaFormServe_* ewaFormSubmit_* ...	The application resumes the request. The server re-invokes the application code attached to the requested document in order to construct a response on-the-fly.
	ewaNetBufferAlloc ewaNetHTTPSend (STATUS_YIELD)	The server allocates and sends a response buffer. The application transmits the data and requests that the server yield the CPU.
ewsRun ...	ewaNetBufferAlloc ewaNetHTTPSend (STATUS_YIELD) ...	The application reschedules the server. The server sends the next buffer.
ewsRun	ewaNetBufferFree ... ewaFree ... ewaNetHTTPEnd	The application reschedules the server. The server releases resources and terminates processing of the request.

The next table illustrates a typical scenario in which the application aborts processing of an incomplete HTTP request.

Application->Server (Abort)	Server -> Application	Description
ewsNetHTTPStart	ewaAlloc ...	Application accepts a new HTTP TCP connection request and informs the EmWeb/Server. The server allocates memory resources for the request.
ewsNetHTTPReceive ...	ewaAlloc ... ewaNetBufferFree ...	Application accepts request data from the network and passes it to the server for processing. The server may allocate memory resources to process the request as needed. Some network buffers received may be released at this point.

Application->Server (Abort)	Server -> Application	Description
ewsNetHTTPAbort	ewaFree ... ewaNetBufferFree ... ewaNetHTTPEnd	Application requests abort of HTTP transaction in progress. Server releases resources and terminates request.

The next table illustrates the on-demand archive loading feature of EmWeb.

Application->Server (Demand Loading)	Server -> Application	Description
ewsDocumentRegister	ewaAlloc ...	Application registers a URL with EmWeb/Server that is not loaded.
ewsNetHTTPStart	ewaAlloc ...	Application accepts a new HTTP TCP connection request and informs the EmWeb/Server. The server allocates memory resources for the request.
ewsNetHTTPReceive ...	ewaAlloc ... ewaNetBufferFree ...	Application accepts request data from the network and passes it to the server for processing. The server may allocate memory resources to process the request as needed. Some network buffers received may be released at this point.
	ewaDocumentFault	The server recognizes the requested URL to be a registered document and notifies the application.
ewsDocumentInstallArchive		The application loads the archive containing the document.
ewsRun	<EMWEB_STRING> <EMWEB_INCLUDE> ewaFormServe_* ewaFormSubmit_* ewaCGIStart_* ewaCGIData_* ...	The server automatically continues processing the request once the document has been loaded. The server invokes the application code attached to the requested document in order to construct a response on-the-fly.
	ewaNetBufferAlloc ... ewaNetHTTPSend (STATUS_OK) ...	The server allocates network buffers as needed, fills them with HTTP response data, and sends them to the application for transmission.

Application->Server (Demand Loading)	Server -> Application	Description

	`ewaFree`
	`...`
	`ewaNetBufferFree`
	`...`
	`ewaNetHTTPEnd`
		The server releases any remaining resources associated with the HTTP transaction and terminates the request. The application closes the TCP connection at this point.

4.3. Porting Guidelines

The EmWeb/Server is distributed as a directory tree of ANSI C files. In order to assure the best possible upgrade path and support from Agranat Systems, most of these files should not be modified by the system integrator.

To port EmWeb/Server to a specific target environment *foo*, a target-specific `config.foo` configuration directory must be created. `config.foo` contains the target-specific configuration files `ew_types.h`, `ew_config.h`, and `config.mak`. These files can be copied from the examples provided in the `config` directory, and modified as appropriate for the target.

To build the server library `obj.foo/ews.a`, use the following command:

```
make CONFIG=foo server
```

> **Note:** The `configure` script may be used on Unix systems to automatically generate the configuration files for the local Unix development environment target. Typing "make" will build the EmWeb/Compiler, the example archives, and the reference Unix port.

4.3.1. Configuration Header Files

The `src/config/ew_types.h` file contains definitions for base C types used throughout the EmWeb/Server. Most of these are straightforward and generally do not require modification under 32-bit CPU architectures. The one definition which may require modification by the system integrator is the preprocessor symbol `EMWEB_ENDIAN` which is defined to either `EMWEB_ENDIAN_BIG` or `EMWEB_ENDIAN_LITTLE`, and reflects the byte-order of the target processor (Intel processors are generally little endian while Motorola processors are generally big endian). The symbol `EMWEB_ARCHIVE_ENDIAN` indicates the byte-order of the archive generated by the EmWeb/Compiler. By default, the compiler generates big-endian archives unless the `--little` command-line argument is present. Note that it is more efficient if the byte-order of the archive matches the byte-order of the target processor.

The `src/config/ew_config.h` file definitions are expected to be modified extensively for porting EmWeb/Server to a particular hardware platform as follows:

EW_CONFIG_HTTP_PROTOCOL

Define the level of protocol conformance desired. Must be set to either HTTP_1_0 or HTTP_1_1.

EW_CONFIG_OPTION_DATE

Define to enable generation of HTTP `Date:` headers. This function requires that the application provide the `ewaDate` function. This is required for protocol conformance, but some environments may simply not be able to determine the time-of-day.

EW_CONFIG_OPTION_EXPIRE

Define to enable generation of HTTP `Expire:` headers. The symbol `EWS_CONFIG_OPTION_DATE` must also be defined. If a dynamic document is requested (i.e. a document containing content defined by the application as it is being served), then an `Expire:` header is generated with the current time. Otherwise, no `Expire:` header is generated. The code size may be reduced by not defining this symbol. This is recommended for proper HTTP caching.

EW_CONFIG_OPTION_LAST_MODIFIED

Define to enable generation of HTTP `Last-modified:` headers. The symbol `EWS_CONFIG_OPTION_DATE` must also be defined. If a static document is requested, then the archive creation date is returned to the Web browser. Otherwise, the current time is returned to the Web browser. The code size may be reduced by not defining this symbol. This is recommended for proper HTTP caching.

EW_CONFIG_OPTION_CONDITIONAL_GET

Define to enable parsing of received `If-Modified-Since:` headers to support conditional get functionality defined in the standard. This is recommended for proper HTTP caching.

EW_CONFIG_OPTION_PRAGMA_NOCACHE

Define to enable generation of "`Pragma: no-cache`" HTTP/1.0 headers for documents containing dynamic content.

EW_CONFIG_OPTION_PERSISTENT

Define to enable persistent connections. This is recommended, especially with HTTP/1.1, for optimum network performance. Persistent connections enable the browser to pipeline multiple HTTP requests over a single TCP/IP connection which was not possible in the original HTTP/1.0 specification.

EW_CONFIG_OPTION_CHUNKED_OUT

HTTP/1.1 only. Define to enable the generation of chunked encoded data during transmission of documents containing dynamic elements. This makes maintaining a persistent connection possible under certain

circumstances and is strongly recommended.

EW_CONFIG_OPTION_CHUNKED_IN
HTTP/1.1 only. Required for protocol conformance. Define to enable parsing of chunked form data received from the browser.

EW_CONFIG_OPTION_METHOD_OPTIONS
Define to enable HTTP/1.1 options method.

EW_CONFIG_OPTION_METHOD_TRACE
Define to enable HTTP/1.1 trace method.

EW_CONFIG_OPTION_CACHE_CONTROL
Define to enable generation of HTTP/1.1 Cache-Control: headers. Recommended for optimum cache control.

EW_CONFIG_OPTION_STRING
Define if the application intends to use the `<EMWEB_STRING>` feature. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_STRING_TYPED
Define to enable typed `EMWEB_STRINGS` (i.e. use of `EMWEB_TYPE` attribute).

EW_CONFIG_OPTION_INCLUDE
Define if the application intends to use the `<EMWEB_INCLUDE>` feature. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FORM
Define if the application intends to use the EmWeb form processing interfaces. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_IMAGEMAP
Define if the application intends to use EmWeb imagemaps. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI
Define if the application intends to use the raw CGI application interfaces. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_LINK
Define if the application intends to use the `link` attribute in `_ACCESS` files. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CLONING
Define if the application intends to use the `ewsDocumentClone` application

interface. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_DEMAND_LOADING
Define if the application intends to use the `ewsDocumentRegister/ewaDocumentFault` application interface. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_DOCUMENT_DATA
Define if the application intends to use the `ewsDocumentData` application interface. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_DOCUMENT_SET_REALM
Define if the application intends to use the `ewsDocumentSetRealm` application interface. Otherwise, code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CLEANUP
Define if the application desires to perform a graceful shutdown of the EmWeb/Server by invoking `ewsShutdown`. In some application environments, graceful shutdown may not be necessary as a system restart may be accomplished by a re-boot of the processor. The code size may be reduced by not defining this symbol

EW_CONFIG_OPTION_SCHED
Define if the application intends to use EmWeb's internal scheduler making use of the `ewsRun` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_SCHED_SUSP_RES
Define if the application intends to use the `ewsSuspend/ewsResume` application interfaces. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_SCHED_FC
Define if the application intends to use the `ewsNetFlowControl/UnFlowControl` application interfaces. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_URL_HOOK
Define if the application intends to use the URL rewriting feature by providing the function `ewaURLHook` interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_AUTH
Define to enable support for authentication through the `ewsAuthRegister` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_AUTH_BASIC

Define to enable support for the HTTP/1.0 basic cookie authentication method. The symbol EW_CONFIG_OPTION_AUTH must also be defined. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_AUTH_MBASIC

Define to enable support for the "manual basic" authentication scheme. The code size may be reduced by not defining this symbol,

EW_CONFIG_OPTION_AUTH_DIGEST

Define to enable support for the HTTP/1.1 digest authentication method. The symbol EW_CONFIG_OPTION_AUTH must also be defined. The code size may be reduced by not defining this symbol.

Note: EmWeb's digest authentication support is derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm. The following legal notice can be found in src/lib/ew_md5c.c:

```
Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
```

EW_CONFIG_OPTION_AUTH_MDIGEST

Define to enable support for the "manual digest" authentication scheme. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_AUTH_DIGEST_M

Define to enable support for the digestRequired parameter used in digest authentication. This protects against forged messages using proper authentication credentials. The symbol EW_CONFIG_OPTION_AUTH_DIGEST must also be defined. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_AUTH_VERIFY
Define to enable support for secondary application-defined authentication. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_RELEASE_UNUSED
Define to include extra code that attempts to release received network buffers as soon as possible if they do not contain information needed for the duration of an HTTP request. Otherwise, if HTTP headers are typically received in multiple network buffers, the EmWeb/Server may hold on to these buffers longer than necessary.

EW_CONFIG_OPTION_FILE
Define to include local file system support. The code size may be reduced by not defining this symbol

EW_CONFIG_OPTION_FILE_GET
Define to include access to local filesystem for GET method requests. This option requires `EW_CONFIG_OPTION_FILE`. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_COMPRESS
Define to include extra code required for decompression of compressed archives.

EW_CONFIG_OPTION_CONTEXT_DATE
Define to enable the `ewsContextDate` application interface. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CONTEXT_PRAGMA
Define to enable the `ewsContextPragma` application interface. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CONTEXT_FROM
Define to enable the `ewsContextFrom` application interface. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CONTEXT_IF_MODIFIED_SINCE
Define to enable the `ewsContextIfModifiedSince` application interface. Furthermore, EmWeb/Server honors the HTTP `If-modified-since:` header for static documents by comparing to the archive creation date. (Note that dynamic documents are always served). The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CONTEXT_REFERER
Define to enable the `ewsContextReferer` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CONTEXT_USER_AGENT
Define to enable the `ewsContextUserAgent` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CONTEXT_HOST
Define to enable the ewsContextHost application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_SEND_REPLY
Define to enable the ewsContextSendReply application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_SERVER_SOFTWARE
Define to enable the `ewsCGIServerSoftware` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_GATEWAY_INTERFACE
Define to enable the `ewsCGIGatewayInterface` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_SERVER_PROTOCOL
Define to enable the `ewsCGIServerProtocol` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_REQUEST_METHOD
Define to enable the `ewsCGIRequestMethod` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_PATH_INFO
Define to enable the `ewsCGIPathInfo` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_SCRIPT_NAME
Define to enable the `ewsCGIScriptName` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_QUERY_STRING
Define to enable the `ewsCGIQueryString` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_CONTENT_TYPE
Define to enable the `ewsCGIContentType` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_CONTENT_LENGTH
Define to enable the `ewsCGIContentLength` application interface. Otherwise,

the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_CGI_CONTENT_ENCODING
Define to enable the `ewsCGIContentEncoding` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_RADIO
Define to enable support for EmWeb HTML form radio buttons. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE
Define to enable support for EmWeb HTML form single option selection boxes. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE
Define to enable support for EmWeb HTML form multiple option selection boxes. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX
Define to enable support for EmWeb HTML form checkbox fields. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_TEXT
Define to enable support for EmWeb HTML form text fields. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_IMAGE
Define to enable support for EmWeb HTML form image inputs. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT
Define to enable support for EmWeb HTML form text fields for unsigned integers. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT
Define to enable support for EmWeb HTML form text fields for signed integers. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_HEX_INT
Define to enable support for EmWeb HTML form text fields for hexadecimal integers. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING
Define to enable support for EmWeb HTML form text fields for hexadecimal octet strings. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP
Define to enable support for EmWeb HTML form text fields for dotted IP addresses. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV
Define to enable support for EmWeb HTML form text fields for DECnet IV addresses. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC
Define to enable support for EmWeb HTML form text fields for IEEE MAC addresses. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC
Define to enable support for EmWeb HTML form text fields for FDDI MAC addresses. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_STD_MAC
Define to enable support for EmWeb HTML form text fields for big or little endian standard MAC addresses. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_OID
Define to enable support for EmWeb HTML form text fields for SNMP object identifiers. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_FIELDTYPE_FILE
Define to enable support for RFC 1867 file upload from browser. This option requires EW_CONFIG_OPTION_FILE. The code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_BROKEN_IMS_EXTRA_DATA
Define to handle non-conformant browsers that place extra data at the end of If-Modified-Since: headers. Otherwise, the code size may be reduced by not defining this symbol.

EW_CONFIG_OPTION_BROKEN_NEED_OPAQUE
Define to handle non-conformant browsers that require optional opaque field in digest authentication headers. Otherwise, the code size may be reduced by not defining this symbol.

EwaNetBuffer
Define the C type used to represent a buffer descriptor.

EWA_NET_BUFFER_NULL
Define the value of a NULL buffer descriptor used to terminate a chain of buffers or indicate no buffers available.

EwaNetHandle
Define the C type used for the application-defined network handle passed to EmWeb/Server in `ewsNetHTTPStart` and available from the request context by `ewsContextNetHandle`.

EwaDocumentHandle
Define the C type used for the application-defined document handle passed to EmWeb/Server in `ewsDocumentClone` or `ewsDocumentRegister` and available from the request context by `ewsContextDocumentHandle`.

EWA_DOCUMENT_HANDLE_NULL
Define the value for a NULL document handle stored in the context for documents that have not been cloned or registered.

EwaAuthHandle
Define the C type used for the application-defined authorization handle passed to EmWeb/Server in `ewsAuthRegister` and available from the request context by `ewsContextAuthHandle`.

EWS_AUTH_HANDLE_NULL
Define the value for a NULL authorization handle stored in the context for requests that have not been authenticated.

EwaAuthNonce
Define the C structure containing parameters used in generating nonce challenges.

EwaCGIHandle
Define the C type used for the application-defined CGI handle passed to EmWeb/Server from `ewaCGIStart_*` and returned to the application in subsequent `waCGIData_*`.

EWA_CGI_HANDLE_NULL
Define the value for a NULL CGI handle stored in the context for CGI

requests.

EWA_TASK_LOCK()
This application-defined macro is invoked by EmWeb/Server to enter a critical region. If the application is preemptive and multiple threads may access EmWeb application interface functions simultaneously, then this macro should be defined in such a way as to disable preemption.

EWA_TASK_UNLOCK()
This application-defined macro is invoked by EmWeb/Server to leave a critical region entered by `EWA_TASK_LOCK()`.

EMWEB_ERROR()
This application-defined macro is invoked by EmWeb/Server to report serious error conditions, usually a result of improper system integration.

EMWEB_WARN()
This application-defined macro is invoked by EmWeb/Server to report an error condition for which recovery is possible.

EMWEB_TRACE()
This application-defined macro is invoked by EmWeb/Server to trace execution for debugging.

EWA_LOG_HOOK
Define to enable the `ewaLogHook()` application interface. Otherwise, the code size may be reduced by not defining this symbol.

EMWEB_SANITY
Define to enable extra code for checking the consistent use of the API and internal data structures. This is strongly recommended during initial porting and debug. However, the code size may be reduced by not defining this symbol.

EMWEB_HAVE_MEMCPY
Define to use application-provided run-time `memcpy` library function. Otherwise, extra server code is included to implement this functionality.

EMWEB_HAVE_MEMSET
Define to use application-provided run-time `memset` library function. Otherwise, extra server code is included to implement this functionality.

EMWEB_HAVE_SPRINTF
Define to use application-provided run-time `sprintf` library function. Otherwise, extra server code is included to handle conversions from integers to strings (%d, %x).

EMWEB_HAVE_STRCPY
Define to use application-provided run-time `strcpy` library function. Otherwise, extra server code is included to implement this functionality.

EMWEB_HAVE_STRLEN
Define to use application-provided run-time `strlen` library function. Otherwise, extra server code is included to implement this functionality.

EMWEB_HAVE_STRCMP
Define to use application-provided run-time `strcmp` library function. Otherwise, extra server code is included to implement this functionality.

EwaFileHandle
Define the C type used for the application-defined file handle used in the local filesystem API.

EWA_FILE_HANDLE_NULL
Define the value for a `NULL` file handle as used by the local filesystem API.

EwaFileName
Define the C structure used to represent an application-defined filename parameter as used by the local filesystem API.

The following macros may be defined to override EmWeb/Server defaults as follows:

EWS_FILE_HASH_SIZE
Default: 256. Information about each unique URL loaded from an archive, registered, or cloned, is maintained in an open hash table for fast lookup. This macro may be defined to set the number of entries in the hash table, where each entry contains a pointer value. Smaller values will typically decrease memory usage while increasing search times.

EWS_REALM_HASH_SIZE
Default: 4. Information about each unique realm defined in loaded archives or by the `ewsDocumentSetRealm()` function is maintained in an open hash table for fast lookup. This macro may be defined to set the number of entries in the hash table, where each entry contains a pointer value. Smaller values will typically decrease memory usage while increasing search times.

EWS_NONCE_QUEUE_SIZE
Default: 4. If digest authentication is used, outstanding nonce values are maintained in a circular queue. This macro may be defined to set the maximum number of outstanding nonce values, where each value will contain approximately 48 bytes of state information plus the size of the application-defined `EwaAuthNonce` structure. Larger values are recommended if one-time nonces are to be used and/or many simultaneous

authenticated requests from different clients are likely.

EWS_HTTP_STATUS_204, EWS_HTTP_STRING_204
Default: "No Content", "\r\n". These macros may be used to override the status line and message body issued for a no content HTTP response. (This status is generated in response to a form submission application function returning NULL, or to an undefined region of an imagemap if no default URL was specified).

EWS_HTTP_STATUS_304, EWS_HTTP_STRING_304
Default: "Not Modified", "\r\n". These macros may be used to override the status line and message body issued for a not modified HTTP response. (This status is generated in response to a conditional GET request if the requested document has not changed since the "If-Modified-Since:" value).

EWS_HTTP_STATUS_400, EWS_HTTP_STRING_400
Default: "Bad Request", "\r\n400 Bad Request\r\n". These macros may be used to override the status line and message body issued for a bad request HTTP response. (This status is generated in response to an HTTP request that could not be parsed).

EWS_HTTP_STATUS_401, EWS_HTTP_STRING_401
Default: "Unauthorized", "\r\n401 Unauthorized\r\n". These macros may be used to override the status line and message body issued for an unauthorized HTTP response. (This status is generated in response to an HTTP request that did not contain proper credentials to access the server).

EWS_HTTP_STATUS_404, EWS_HTTP_STRING_404
Default: "Not Found", "\r\n404 Not Found\r\n". These macros may be used to override the status line and message body issued for a not found HTTP response. (This status is generated in response to an HTTP requests for an unknown or hidden URL).

EWS_HTTP_STATUS_411, EWS_HTTP_STRING_411
Default: "Length Required", "\r\n411 Length Required\r\n". These macros may be used to override the status line and message body issued for an internal error HTTP response. (This status is generated in response to an HTTP/1.1 request containing chunk encoded form data with EW_CONFIG_OPTION_CHUNKED_IN disabled).

EWS_HTTP_STATUS_500, EWS_HTTP_STRING_500
Default: "Internal Error", "\r\n500 Internal Error\r\n". These macros may be used to override the status line and message body issued for an internal error HTTP response. (This status is generated in response to an HTTP request resulting in a detectable internal error. This should not ever occur.)

EWS_HTTP_STATUS_501, EWS_HTTP_STRING_501
Default: "Not Implemented", "\r\n501 Not Implemented\r\n". These macros may be used to override the status line and message body issued for a not implemented HTTP response. (This status is generated in response to an HTTP request containing a method not supported by the URL).

EWS_HTTP_STATUS_503, EWS_HTTP_STRING_503
Default: "Service Unavailable", "\r\n503 Service Unavailable\r\n". These macros may be used to override the status line and message body issued for a service unavailable HTTP response. (This status is generated in response to an HTTP request for a registered and unloaded document when the application-specific ewaDocumentFault() function aborts the request rather than loading the archive containing the URL).

4.3.2. Application-Provided Functions

The following functions must be provided by the application:

- System functions

. ewaAlloc
. ewaFree

- Network functions

. ewaNetBufferAlloc
. ewaNetBufferFree
. ewaNetBufferNextGet
. ewaNetBufferNextSet
. ewaNetBufferLengthGet
. ewaNetBufferLengthSet
. ewaNetBufferDataGet
. ewaNetBufferDataSet
. ewaNetHTTPSend
. ewaNetHTTPEnd
. ewaNetHTTPCleanup
. ewaNetLocalHostName

The following functions may be provided by the application if certain optional EmWeb/Server features are desired.

- System functions

. ewaDate
. ewaLogHook

- Document functions

  . ewaDocumentFault
  . ewaURLHook

- Authentication functions

  . ewaAuthRealmQualifier

- Digest Authentication functions

  . ewaAuthNonceCreate
  . ewaAuthNonceCheck

- Manual Authentication functions

  . ewaAuthCheckBasic
  . ewaAuthCheckDigest
  . ewaAuthVerifySecurity

4.4. Memory Requirements

The following table illustrates the approximate incremental executable image size of the EmWeb/Server for various configuration options enabled compiled using GNU gcc-2.7.0 with -O2 for the Intel 486, the Intel i960, and the Motorola MC68000 architectures. These values are subject to change without notice.

Incremental Option	i486	i960	MC68000
Base EmWeb/Server functionality	7,380	7,496	6,238
<EMWEB_STRING>	180	216	184
Typed <EMWEB_STRING>s	464	464	454
<EMWEB_INCLUDE> *(requires some additional code from <EMWEB_STRING>).*	384	400	312
Base EmWeb HTML form support	3,124	3,168	2,548
Imagemap support	288	320	272
Base raw CGI support	640	656	578
URL Link support	16	32	28
Document cloning	400	400	314
Document on-demand loading	492	520	440

Incremental Option	i486	i960	MC68000
Document data access	64	72	74
Graceful server shutdown	272	264	220
Request scheduling (yield CPU)	656	624	444
Proxy (ewsSuspend/ewsResume) *(requires request scheduling)*	320	296	242
Network flow control *(requires request scheduling)*	368	288	264
URL rewriting (ewaURLHook)	304	208	164
Base filesystem support	176	224	170
GET/HEAD from local filesystem *(Requires base filesystem sup.)*	1,876	1,480	1,326
Base authentication support	1,168	1,032	934
Basic authentication	952	1,056	824
Digest authentication	5,547	6,732	5,366
Client message digest verification *(requires digest authentication)*	1,335	1,016	742
Manual basic authentication *(requires basic authentication)*	608	472	482
Manual basic decode *(requires manual basic)*	336	352	198
Manual digest authentication *(requires digest authentication)*	384	368	366
Manual authentication verification	144	144	128
Run-time document realm override	0	0	0
Base HTTP/1.1 support	1,284	1,160	972
Persistent connections	883	616	614
Generate chunked encoding *(requires HTTP/1.1)*	828	784	632
Parse chunked encoding *(requires HTTP/1.1)*	855	552	468
OPTIONS method	250	240	192

Incremental Option	i486	I960	MC68000
TRACE method	288	288	204
Generate Cache-Control headers	122	128	96
Generate Date headers (ewaDate)	41	64	32
Generate Expire headers (ewaDate)	76	176	60
Generate Last-Modified headers	82	112	76
Conditional get support	239	280	248
Generate Pragma headers	35	48	34
Early release of network buffers	224	240	208
Compression	525	584	462
Parse Date headers	64	56	72
Parse Pragma headers	75	64	82
Parse From headers	73	72	82
Parse If-Modified-Since headers	64	64	72
Parse Referer headers	76	64	86
Parse User-Agent headers	79	104	90
Parse Host headers	169	128	124
ewsContextSendReply	626	632	494
ewsCGIServerSoftware	31	40	30
ewsCGIGatewayInterface	24	32	22
ewsCGIServerProtocol	64	56	68
ewsCGIRequestMethod	64	64	68
ewsCGIPathInfo	16	16	16
ewsCGIScriptName	16	16	16
ewsCGIQueryString	64	64	68
ewsCGIContentType	80	64	72
ewsCGIContentLength	16	16	16
ewsCGIContentEncoding	64	64	72

Incremental Option	i486	i960	MC68000
Form radio buttons	328	304	254
Form single option select boxes	57	64	62
Form multiple option select boxes	128	144	124
Form checkboxes	96	96	92
Form text fields	307	352	256
Form image input fields	93	104	86
Form text fields - unsigned integer	0	0	0
Form text fields - signed integer *(requires some code from unsigned integer)*	98	128	90
Form text fields - hexadecimal int *(requires some code from unsigned integer)*	164	184	166
Form text fields - hex string *(requires some code from unsigned integer)*	706	704	550
Form text fields - dotted IP *(requires some code from unsigned integer)*	354	416	310
Form text fields - DECNet IV *(requires some code from unsigned integer)*	272	352	280
Form text fields - IEEE MAC	144	160	136
Form text fields - FDDI MAC *(requires some code from IEEE MAC)*	48	48	44
Form text fields - Standard MAC *(requires some code from IEEE MAC)*	160	192	168
Form text fields - SNMP Object ID	400	400	340
Form file fileds - file upload *(requires base filesystem support)*	4,372	4,232	3,394
Broken If-Modified-Since Extra	84	128	124

Incremental Option	i486	i960	MC68000
Broken digest need opaque	16	16	16
Internal sprintf library	656	736	582
Internal strcpy library	112	336	46
Internal strcmp library	128	184	52
Internal strlen library	-160	384	162
Internal memcpy library	160	80	-8
Internal memset library	16	16	-2
Extra sanity checking code	1,248	1,080	1,008
Error reporting	2,498	2,984	2,358
Warning reporting	2,527	2,888	2,350
Debug tracing	1,686	2,392	1,764
Logging hook	704	1,056	778
FULL CONFIGURATION	52,577	55,348	44,740

Note that some of the configurable options listed in the table above depend on pieces of code included from previous options as indicated. The table shows the incremental requirements as each option is added from top to bottom. Therefore, the executable image size for an arbitrary set of options may be different than the sum of the values shown above.

Additional archive memory requirements include

- Compressed file contents and embedded application object code (controlled by application)
- 120 bytes per archive
- 36 bytes per document in archive (plus null-terminated URL string)
- 12 bytes per <EMWEB_STRING>, <EMWEB_INCLUDE>, CGI script, or form field
- null terminated strings for authorization realms, mime types, creation date, etc.
- compression dictionary (if compression used).

Additional run-time memory requirements include:

- Network buffers (controlled by application). EmWeb/Server typically holds on to received buffers containing HTTP headers of interest for the duration of a request, and typically allocates only one or two buffers before sending them to the application for transmission.

- 1,028 - 1,096 bytes total global EmWeb state (using defaults for hash table sizes which may be overridden by application)

- 176 - 636 bytes per HTTP request (plus 48-52 bytes for first served document, plus 48 bytes for decompression context if compressed)

- 48-52 bytes per nested <EMWEB_INCLUDE> document (plus 48 bytes for decompression context if compressed)

- 40-64 bytes per URL in archive (including clones and registered documents)

- 20-44 bytes per authorization realm plus null-terminated realm name string

- 20-28 bytes per authorization entry plus Base64 or MD5 encoding of user/password

5. Conformance

The EmWeb/Server implements the HTTP/1.1 and HTTP/1.0 protocols, and MD5 Digest Authentication.

The EmWeb/Server raw CGI capabilities support the functionality of CGI/1.1, but the interfaces are non-standard as they are optimized for the EmWeb/Server environment. In addition, some CGI/1.1 interfaces depend upon the specific integration of EmWeb/Server into the application environment.

The EmWeb/Compiler supports HTML/2.0 Forms with RFC 1867 file upload extensions.

6. Release History

Release 1.0 (12/23/96)	General Availability
Release 1.1 (02/16/97)	Added network flow-control interfaces
Release 1.2 (02/26/97)	Ported to VxWorks
Release 1.3 (03/07/97)	Ported to pSOS
Release 2.0 (04/14/97)	Added HTTP/1.1 support Added support for typed EMWEB_STRINGs Added EMWEB_STATIC cache-control override Added support for remote authentication databases
Release 2.1 (06/20/97)	Added native filesystem support Added RFC 1867 (file upload from browser) support Added support for EMWEB_TYPE=OBJECT_ID Added support for SNMP Research <mibobj> tags Added ewaNetHTTPCleanup() interfaces

7. References

[1] T. Berners-Lee and D. Connolly. "Hypertext Markup Language - 2.0." RFC 1866, MIT/W3C, November 1995.T. Berners-Lee and D. Connolly. "Hypertext Markup Language - 2.0." RFC 1866, MIT/W3C, November 1995.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May, 1996.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[4] R. Braden, "Requirements for Internet Hosts -- Application and Support", RFC1123, October 1989.

[5] M. Horton, "Standard for Interchange of USENET Messages", RFC850, June 1983.

[6] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart. "An Extension to HTTP: Digest Access Authentication", RFC 2069, January 1997.

[7] E. Nebel, L. Masinter. "Form-based File Upload in HTML" RFC 1867, Xerox Corporation, November 1995.

A. EmWeb Application Interface Header Files

This appendix is intended for reference only and is not part of this specification. The C header files listed here are part of the standard EmWeb source distribution and may be updated from time to time.

A-1. Configuration

These sample configuration header files are provided in the EmWeb source distribution. It is expected that these files will be modified by the system integrator in order to port the EmWeb/Server to the target environment. No other files in the EmWeb distribution should be modified in any way.

A-1.1. src/config/ew_types.h

This files contains definitions for base C types used throughout the EmWeb product. In most environments, this file will not need to be modified except possibly for the definition of EMWEB_ENDIAN and EMWEB_ARCHIVE_ENDIAN. However, for some compilers, it may be necessary for the system integrator to modify additional definitions here.

```
/*
 * ew_types.h,v 1.20 1997/03/31 19:14:25 lawrence Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb common types
 *
 */
#ifndef _EW_TYPES_H
#define _EW_TYPES_H

/*
```

```
 * C-TYPES
 * The following basic C-types are defined (see ew_config.h)
 */
#ifndef INT8
#   define INT8 signed char
#endif

#ifndef UINT8
#   define UINT8 unsigned char
#endif

#ifndef INT16
#   define INT16 short int
#endif

#ifndef INT32
#   define INT32 long int
#endif

#ifndef P_INT                    /* an integer the same size as a pointer */
#   define P_INT int
#endif

typedef INT8            int8;           /* signed eight bit integer */
typedef INT16           int16;          /* signed sixteen bit integer */
typedef INT32           int32;          /* signed thirty-two bit integer */
typedef UINT8           uint8;          /* unsigned eight bit integer */
typedef unsigned INT16  uint16;         /* unsigned sixteen bit integer */
typedef unsigned INT32  uint32;         /* unsigned thirty-two bit integer */

/*
 * Volatiles for memory shared with devices
 */
typedef volatile int8   vint8;          /* volatile signed eight bits */
typedef volatile int16  vint16;         /* volatile signed sixteen bits */
typedef volatile int32  vint32;         /* volatile signed thirty-two bits */
typedef volatile uint8  vuint8;         /* volatile signed eight bits */
typedef volatile uint16 vuint16;        /* volatile unsigned sixteen bits */
typedef volatile uint32 vuint32;        /* volatile unsigned thirty-two bits */

/*
 * FAST TYPES
 * The following types offer the most efficient compilation for optimized speed
 * These are never assumed to be more than 16 bits.
 */
typedef int             sintf;          /* signed integer */
typedef unsigned        uintf;          /* unsigned integer */

/*
 * BOOLEAN
 * The boolean type is used primarily as a function return type.  Note that
 * a default representation of TRUE and FALSE are defined.  However, care
 * should be taken to avoid comparing against these values.  For most compilers,
 * TRUE can be any non-zero integer while FALSE is defined as zero.
```

```
 */
#define TRUE            (0 == 0)
#define FALSE           (!TRUE)

typedef uintf           boolean;        /* true or false (non-zero or zero) */

/*
 * POINTER INTEGER
 * The pointer integer is an unsigned integer large enough to accommodate a
 * machine pointer.  This is used primarily for generic pointer arithmetic.
 */
typedef P_INT           pint;           /* unsigned integer for pointer */

/*
 * NULL POINTER
 */
#ifndef NULL
#define NULL            (0)
#endif /* NULL */

/*
 * ENDIAN (don't change these)
 */
#define EMWEB_ENDIAN_BIG        1234
#define EMWEB_ENDIAN_LITTLE     4321

typedef enum Endian_e
{
  ewBigEndian    = EMWEB_ENDIAN_BIG,
  ewLittleEndian = EMWEB_ENDIAN_LITTLE
} Endian;

#if 0
/*
 * EMWEB_ENDIAN - Define as endian-ness of target architecture
 */
#ifndef EMWEB_ENDIAN
#  ifdef HAVE_CONFIG_H
#    include "config.h"
#    ifdef WORDS_BIGENDIAN
#      define EMWEB_ENDIAN EMWEB_ENDIAN_BIG
#    else
#      define EMWEB_ENDIAN EMWEB_ENDIAN_LITTLE
#    endif
#  else
#    pragma error "Define EMWEB_ENDIAN for target"
#  endif
#else
#  pragma error "Define EMWEB_ENDIAN for target"
#endif /* HAVE_CONFIG_H */
#endif /* !TBD!  */

/*
 * EMWEB_ARCHIVE_ENDIAN
```

```
 * Define endian for target archive as generated by EmWeb/Compiler
 */
#define EMWEB_ARCHIVE_ENDIAN    EMWEB_ENDIAN_BIG

/*
 * COMPILER COMPATIBILITY
 *
 * __BEGIN_DECLS precedes external definitions, __END_DECLS follows external
 * definitions.  These could be defined as 'extern "C" {' and '}' respectively
s * for GNU C++ compilers, for example.
 */
#ifdef __cplusplus
#define __BEGIN_DECLS extern "C" {
#define __END_DECLS }
#else /* __cplusplus */
#define __BEGIN_DECLS
#define __END_DECLS
#endif /* __cplusplus */

/*
 * PROTOTYPES
 */
__BEGIN_DECLS
__END_DECLS


#endif /* _EW_TYPES_H */
```

A-1.2. src/config/ew_config.h

This file must be modified by the system integrator to taylor EmWeb/Server to the target environment.

```
/*
 * ew_config.h,v 1.77.2.3 1997/06/15 19:22:31 ian Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
```

```
 *
 * http://www.agranat.com/
 *
 * EmWeb common configuration
 *
 */
#ifndef _EW_CONFIG_H
#define _EW_CONFIG_H

/*
 * Types for sized integers: INT8 INT16 INT32 P_INT
 *         These are guesses as to the correct types to produce the
 *         required sizes for your system.
 *         The configure script will attempt to set these correctly
 *         for the compiler platform; you should set when you copy this
 *         file for each target.
 */
#ifndef INT8
#   define INT8 signed char
#endif

#ifndef INT16
#   define INT16 short int
#endif

#ifndef INT32
#   define INT32 int
#endif

#ifndef P_INT                    /* an integer the same size as a pointer */
#   define P_INT int
#endif

#include "ew_types.h"

/*
 * INCLUDES
 * Any system includes needed by application environment go here
 */
extern void application_tick();
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <signal.h>

/*
 * HTTP PROTOCOL VERSION
 */
#       define HTTP_1_0 1000    /* DO NOT MODIFY */
#       define HTTP_1_1 1001    /* DO NOT MODIFY */
/*
 * Select desired conformance level
 */
```

```
#define EW_CONFIG_HTTP_PROTOCOL HTTP_1_1

/*
 * PROTOCOL OPTIONS (>= HTTP/1.0)
 *   (R) Required, but some embedded systems don't have time-of-day
 *   (O) Optional, but strongly recommended
 */
#define EW_CONFIG_OPTION_DATE            /* (R) Generate Date: */
#define EW_CONFIG_OPTION_EXPIRE          /* (O) Generate Expire: */
#define EW_CONFIG_OPTION_LAST_MODIFIED   /* (O) Generate Last-modified: */
#define EW_CONFIG_OPTION_CONDITIONAL_GET/* (O) Parse If-Modified-Since: */
#define EW_CONFIG_OPTION_PRAGMA_NOCACHE /* (O) Generate Pragma: no-cache */

/*
 * PROTOCOL OPTIONS (>= HTTP/1.1)
 *   (O) Optional, but strongly recommended
 *   (*) May be used with HTTP/1.0
 */
#define EW_CONFIG_OPTION_PERSISTENT      /* (O*) persistent connections */
#define EW_CONFIG_OPTION_CHUNKED_OUT     /* (O) respond with chunked encoding */
#define EW_CONFIG_OPTION_CHUNKED_IN      /* (O) parse chunked encoding */
#define EW_CONFIG_OPTION_METHOD_OPTIONS /* (O*) support OPTIONS method */
#define EW_CONFIG_OPTION_METHOD_TRACE    /* (O*) support TRACE method */
#undef EW_CONFIG_OPTION_METHOD_PUT      /* (*) support PUT operations */
#undef EW_CONFIG_OPTION_METHOD_DELETE   /* (*) support DELETE operations */
#define EW_CONFIG_OPTION_CACHE_CONTROL   /* (O*) Generate Cache-Control: */

/*
 * CONFIGURATION OPTIONS
 *
 * The EmWeb/Server may be customized by the application developer to balance
 * between functionality v.s. memory and CPU requirements.  Each of the options
 * below may be selected (#define) or unselected (#undef).
 */
#define EW_CONFIG_OPTION_STRING          /* emweb_string support */
#define EW_CONFIG_OPTION_STRING_TYPED    /* typed emweb_string support */
#define EW_CONFIG_OPTION_INCLUDE         /* emweb_include support */
#define EW_CONFIG_OPTION_FORM            /* form support */
#define EW_CONFIG_OPTION_IMAGEMAP        /* support for imagemaps */
#define EW_CONFIG_OPTION_CGI             /* raw CGI support */
#define EW_CONFIG_OPTION_LINK            /* link node support */
#define EW_CONFIG_OPTION_CLONING         /* URL cloning support */
#define EW_CONFIG_OPTION_DEMAND_LOADING /* demand document loading support */
#define EW_CONFIG_OPTION_DOCUMENT_DATA  /* direct archive data access support */
#define EW_CONFIG_OPTION_DOCUMENT_SET_REALM /* run-time realm assignment sup. */
#define EW_CONFIG_OPTION_CLEANUP         /* graceful cleanup support */
#define EW_CONFIG_OPTION_SCHED           /* request scheduling support */
#define EW_CONFIG_OPTION_SCHED_SUSP_RES /* suspend/resume request support */
#define EW_CONFIG_OPTION_SCHED_FC        /* flow control support */
#define EW_CONFIG_OPTION_URL_HOOK        /* request URL rewriting hook */
#define EW_CONFIG_OPTION_AUTH            /* authorization support */
#define EW_CONFIG_OPTION_AUTH_BASIC      /* basic authorization support */
#defin  EW_CONFIG_OPTION_AUTH_DIGEST     /* digest authorization support */
#define EW_CONFIG_OPTION_AUTH_DIGEST_M  /* ", verify client message digest */
```

```
#define EW_CONFIG_OPTION_AUTH_MBASIC    /* "manual" basic authentication */
#define EW_CONFIG_OPTION_AUTH_MBASIC_DECODE /* do base64 decode for MBASIC */
#define EW_CONFIG_OPTION_AUTH_MDIGEST   /* "manual" digest authentication */
#define EW_CONFIG_OPTION_AUTH_VERIFY    /* manual verification of client */
#define EW_CONFIG_OPTION_COMPRESS       /* archive decompression support */
#define EW_CONFIG_OPTION_RELEASE_UNUSED /* Early release of unused buffers */
#define EW_CONFIG_OPTION_FILE           /* local filesystem support */
#define EW_CONFIG_OPTION_FILE_GET       /* local filesystem GET method */
#define EW_CONFIG_OPTION_FILE_DELETE    /* local filesystem DELETE method */
#define EW_CONFIG_OPTION_FILE_PUT       /* local filesystem PUT method */

/*
 * Options to include support for retrieving HTTP request headers
 */
#define EW_CONFIG_OPTION_CONTEXT_DATE
#define EW_CONFIG_OPTION_CONTEXT_PRAGMA
#define EW_CONFIG_OPTION_CONTEXT_FROM
#define EW_CONFIG_OPTION_CONTEXT_IF_MODIFIED_SINCE
#define EW_CONFIG_OPTION_CONTEXT_REFERER
#define EW_CONFIG_OPTION_CONTEXT_USER_AGENT
#define EW_CONFIG_OPTION_CONTEXT_HOST

/*
 * Option for serving document to browser in response to form or CGI
 */
#define EW_CONFIG_OPTION_CONTEXT_SEND_REPLY

/*
 * Options to include support for CGI access functions
 */
#define EW_CONFIG_OPTION_CGI_SERVER_SOFTWARE
#define EW_CONFIG_OPTION_CGI_GATEWAY_INTERFACE
#define EW_CONFIG_OPTION_CGI_SERVER_PROTOCOL
#define EW_CONFIG_OPTION_CGI_REQUEST_METHOD
#define EW_CONFIG_OPTION_CGI_PATH_INFO
#define EW_CONFIG_OPTION_CGI_SCRIPT_NAME
#define EW_CONFIG_OPTION_CGI_QUERY_STRING
#define EW_CONFIG_OPTION_CGI_CONTENT_TYPE
#define EW_CONFIG_OPTION_CGI_CONTENT_LENGTH
#define EW_CONFIG_OPTION_CGI_CONTENT_ENCODING

/*
 * Options to handle certain form field types (EW_CONFIG_OPTION_FORM must be
 * defined for these to take effect).
 */
#define EW_CONFIG_OPTION_FIELDTYPE_RADIO
#define EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE
#define EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE
#define EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX
#define EW_CONFIG_OPTION_FIELDTYPE_TEXT
#define EW_CONFIG_OPTION_FIELDTYPE_IMAGE
#define EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT
#defin  EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT
#define EW_CONFIG_OPTION_FIELDTYPE_HEX_INT
```

```
#define EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING
#define EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP
#define EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV
#define EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC
#define EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC
#define EW_CONFIG_OPTION_FIELDTYPE_STD_MAC
#define EW_CONFIG_OPTION_FIELDTYPE_OID
#define EW_CONFIG_OPTION_FIELDTYPE_FILE

/*
 * Force use of the buffer early release option if support for
 * file input field - needed so we don't buffer the entire
 * incoming file.
 */

#ifdef  EW_CONFIG_OPTION_FIELDTYPE_FILE
#ifndef EW_CONFIG_OPTION_RELEASE_UNUSED
#define EW_CONFIG_OPTION_RELEASE_UNUSED
#endif /* EW_CONFIG_OPTION_RELEASE_UNUSED */
#endif /* EW_CONFIG_OPTION_FIELDTYPE_FILE */

/*
 * Options to handle non-conformant browsers
 *
 * BROKEN_IMS_EXTRA_DATA - Several recent versions of Netscape incorrectly
 * add extra information at the end of an If-Modified-Since: header in the
 * form "; (size = n)".  This is in violation of the HTTP specification and
 * causes EmWeb/Server to effectively ignore the header.  Since so many
 * browsers exhibit this behaviour, this work-around is provided to ignore
 * extra characters after the date string.
 */
#define EW_CONFIG_OPTION_BROKEN_IMS_EXTRA_DATA

/*
 *
 * BROKEN_NEED_OPAQUE - NCSA_Mosaic/2.7b5 and Spyglass_Mosaic/2.11
 * incorrectly require the server to generate an opaque parameter
 * in WWW-Authenticate: headers using Digest authentication.
 */
#define EW_CONFIG_OPTION_BROKEN_NEED_OPAQUE

/*
 * SANITY CHECKING
 *
 * Define EMWEB_SANITY to include extra sanity checking code during initial
 * integration and debugging.  Code size may be reduced by not defining this.
 */
#define EMWEB_SANITY

/*
 * PORT-SPECIFIC DEFINITIONS
 */
#define ewaAlloc malloc
#define ewaFree free
```

```
/*
 * TRACING AND LOGGING
 *
 * The following macros provide tracing and logging facilities using ANSI
 * printf() style arguments.
 */
#define EMWEB_ERROR(x)  printf x
#define EMWEB_WARN(x)   printf x
#define EMWEB_TRACE(x)  /*printf x*/
#define EWA_LOG_HOOK    /* define to include ewaLogHook() interface */

/*
 * NETWORK BUFFERS
 * The network buffer structure is defined by the application and used
 * by the EmWeb/Server.  A network buffer could be an index into an array,
 * a pointer to a buffer descriptor, etc.  EmWeb/Server imposes only the
 * following requirements on the application's buffer implementation:
 *
 *   o Given a buffer, a pointer to the start of data may be determined.
 *   o Given a buffer, the number of bytes of data may be determined or
 *     specified.  The EmWeb/Server will never attempt to increase the
 *     size of a buffer.
 *   o Given a buffer, the next buffer on a singly linked list may be
 *     determined or specified.
 *
 * The EwaNetBuffer structure is used by EmWeb/Server to represents a buffer
 * descriptor.  The value EWA_NET_BUFFER_NULL indicates a NULL buffer used to
 * terminate a buffer chain or as a return value from EwaNetBufferAlloc()
 * indicating no buffers available.
 */
typedef struct EwaNetBuffer_s
  {
    struct EwaNetBuffer_s       * next;
    uint8                       * realdata;
    uint8                       * data;
    uintf                       length;
  } EwaNetBuffer_t, * EwaNetBuffer;

#define EWA_NET_BUFFER_NULL     ((EwaNetBuffer) NULL)

/* Buffer primitives - these may be macros or subroutines */

#define ewaNetBufferLengthGet(buffer)           (buffer)->length
#define ewaNetBufferLengthSet(buffer, len)      (buffer)->length = len
#define ewaNetBufferDataGet(buffer)             (buffer)->data
#define ewaNetBufferDataSet(buffer, datap)      (buffer)->data = datap
#define ewaNetBufferNextGet(buffer)             (buffer)->next
#define ewaNetBufferNextSet(buffer, nxt)        (buffer)->next = nxt

/*
 * Application-specific Network handle
 */
typedef struct EwaNetHandle_s EwaNetHandle_t, *EwaNetHandle;
```

```
#define EWA_NET_HANDLE_NULL     ((EwaNetHandle) NULL)

/*
 * Application-specific network handle cleanup function (or macro)
 * Invoked by EmWeb/Server after each processing each HTTP request
 * before invoking ewaNetHTTPEnd().
 */
#ifndef EW_CONFIG_OPTION_FILE_GET
#define ewaNetHTTPCleanup( handle ) /* no cleanup necessary */
#endif

/*
 * Application-specific Document handle
 */
typedef void * EwaDocumentHandle;
#define EWA_DOCUMENT_HANDLE_NULL        ((EwaDocumentHandle) NULL)

#ifdef EW_CONFIG_OPTION_AUTH
/*
 * AUTHORIZATION
 *
 * The application may attach an application-specific handle to an entry
 * in the authentication database.  The C-type for this handle is defined
 * by the application below.
 */
typedef void * EwaAuthHandle;           /* user handle to ID entry */
#define EWA_AUTH_HANDLE_NULL    ((EwaAuthHandle) NULL)

#ifdef EW_CONFIG_OPTION_AUTH_DIGEST
/*
 * Application-specific nonce parameters - these are used for Digest
 * authentication to generate one-time challenges.  Recommended values
 * include the client's IP address (derived from the network handle),
 * a time-stamp, a server-side secret value, and other random bits and
 * pieces.  The EmWeb/Server generates an MD5 hash on these values (hiding
 * their semantics) to create the nonce value sent as challenges.
 */
typedef struct EwaAuthNonce_s
  {
    uint32      client_ip;
    uint32      timestamp;
    uint32      up_counter;
#   define      EWA_AUTH_SECRET_SIZE 8
    char        secret[EWA_AUTH_SECRET_SIZE];
  } EwaAuthNonce;

#endif /* EW_CONFIG_OPTION_AUTH_DIGEST */

#endif /* EW_CONFIG_OPTION_AUTH */

/*
 * RAW CGI INTERFACE
 *
 * The application may attach an application-specific handle to a raw CGI
```

```
 * request.  The C-type for this handle is defined by the application below.
 */
typedef void * EwaCGIHandle;
#define EWA_CGI_HANDLE_NULL     ((EwaCGIHandle) NULL)

/*
 * LOCKING
 *
 * The following macros must be defined to disable and enable task preemption
 * to protect critical regions if more than one pre-emptive task may access
 * EmWeb/Server API functions at the same time.
 */
#define EWA_TASK_LOCK()
#define EWA_TASK_UNLOCK()

/*
 * GLOBAL STATE
 */
typedef struct EwsState_s *EwsStateP;
/*
 * The ews_state is a pointer to global state information used internally
 * by EmWeb/Server.  In some environements, it may be desirable to maintain
 * multiple threads of EmWeb/Server within a single memory.  To support this
 * functionality, ews_state may be overridden by a macro that expands at
 * run-time to different values.  The only requirement is that the macro
 * returns type (EwsStateP), and that the address of the pointer (&ews_state)
 * can be determined.  For example:
 *
 * #define ews_state (*((EwsStateP*)someFunction(someArgument)))
 */

/*
 * LIBRARIES
 *
 * The EmWeb/Server implementation uses the following standard C library
 * functions:
 *
 *      sprintf - for converting decimal integers to strings, needed to
 *                generate Content-Length: HTTP headers.
 *
 *      strcpy  - for copying null-terminated strings between the application
 *                and the EmWeb/Server (usually URL names).
 *
 *      strcmp  - for comparing null-terminated strings (comparing URL names).
 *
 *      strlen  - get size of string
 *
 * Most embedded environments have these functions available in libraries.
 * However, EmWeb/Server can be configured to provide its own internal
 * implementations for this purpose.  (Note that a general sprintf
 * implementation is not provided.  Instead, only a conversion routine from
 * long integers to strings is needed.
 *
 * The following macros should be defined (#define) if the corresponding
```

```
 * function is provided by the application environment.
 */
#undef EMWEB_HAVE_SPRINTF
#undef EMWEB_HAVE_STRCPY
#undef EMWEB_HAVE_STRCMP
#undef EMWEB_HAVE_STRLEN
#undef EMWEB_HAVE_MEMCPY
#undef EMWEB_HAVE_MEMSET
#undef EMWEB_HAVE_STRCHR
/* If you have 'index' but not 'strchr', define EMWEB_HAVE_STRCHR and
 *   #define EMWEB_STRCHR(s,c) index(s,c)
 */

#ifdef EW_CONFIG_OPTION_FILE
/*
 * SERVER FILE SUPPORT
 *
 * For support of a server side file system.
 */

typedef struct EwaFileHandle_s *EwaFileHandle;
#define EWA_FILE_HANDLE_NULL ((EwaFileHandle) NULL)

#if defined( EW_CONFIG_OPTION_FILE_GET ) \
 || defined( EW_CONFIG_OPTION_FILE_PUT ) \
 || defined( EW_CONFIG_OPTION_FILE_DELETE )
typedef void *EwaFileName;        /* unused for now */
#endif /* EW_CONFIG_OPTION_FILE_xxx */
#endif /* EW_CONFIG_OPTION_FILE */

#endif /* _EW_CONFIG_H */
```

A-2. Common Header Files

The header files provided here may be included by application software and define the application interfaces to the EmWeb/Server.

A-2.1. src/include/ews_api.h

This is a master include file for EmWeb/Server which, in turn, includes all other application interface header files. We recommend that applications include this file for ease of integration with future releases.

```
/*
 * ews_api.h,v 1.12 1997/05/21 21:55:43 ian Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
```

```
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server Application Interface
 *
 */
#ifndef _EWS_API_H
#define _EWS_API_H

#include "ew_types.h"   /* application-defined generic types */
#include "ew_config.h"  /* application-defined configuration */

#include "ews_def.h"    /* general interface definitions */
#include "ews_sys.h"    /* system interfaces */
#include "ews_net.h"    /* network interfaces */
#include "ews_doc.h"    /* document interfaces */
#include "ews_auth.h"   /* authentication interfaces */
#include "ews_cgi.h"    /* CGI interfaces */
#include "ews_ctxt.h"   /* context access interfaces */

#endif /* _EWS_API_H */
```

A-2.2. src/include/ews_def.h

This file contains general definitions used throughout the EmWeb/Server application interface.

```
/*
 * ews_def.h,v 1.26 1997/05/25 23:30:32 giusti Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
```

```
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server public definitions
 *
 */
#ifndef _EWS_DEF_H
#define _EWS_DEF_H

typedef const char *EwsConstCharP;

/*
 * Status codes returned to the application by EmWeb/Server
 */
typedef enum EwsStatus_e
  {
    EWS_STATUS_OK,
    EWS_STATUS_BAD_MAGIC,
    EWS_STATUS_BAD_VERSION,
    EWS_STATUS_ALREADY_EXISTS,
    EWS_STATUS_NO_RESOURCES,
    EWS_STATUS_IN_USE,
    EWS_STATUS_NOT_REGISTERED,
    EWS_STATUS_NOT_CLONED,
    EWS_STATUS_NOT_FOUND,
    EWS_STATUS_AUTH_FAILED,
    EWS_STATUS_BAD_STATE,
    EWS_STATUS_BAD_REALM,
    EWS_STATUS_FATAL_ERROR,
    EWS_STATUS_ABORTED
  } EwsStatus;

/*
 * Status codes returned to EmWeb/Server by the application
 */
typedef enum EwaStatus_e
  {
    EWA_STATUS_OK,

#   ifdef EW_CONFIG_OPTION_SCHED
    EWA_STATUS_OK_YIELD,
#   endif

    EWA_STATUS_ERROR

  } EwaStatus;

/*
```

```
 * EmWeb/Server request context handle
 */
typedef struct EwsContext_s * EwsContext;
#define EWS_CONTEXT_NULL        ((EwsContext) NULL)

#ifdef EWA_LOG_HOOK
/*
 * Status codes used for logging requests
 */
typedef enum EwsLogStatus_e
  {
    /*
     * 200 Request accepted
     */
    EWS_LOG_STATUS_OK,

    /*
     * Request dispositions (after successful request)
     */
    EWS_LOG_STATUS_NO_CONTENT,              /* 204 no-op form or imagemap */
    EWS_LOG_STATUS_MOVED_PERMANENTLY,       /* 301 link */
    EWS_LOG_STATUS_MOVED_TEMPORARILY,       /* 302 redirect */
    EWS_LOG_STATUS_SEE_OTHER,               /* 303 see other */
    EWS_LOG_STATUS_NOT_MODIFIED,            /* 304 not modified since */

    /*
     * 401 Unauthorized
     */
    EWS_LOG_STATUS_AUTH_FAILED,             /* authorization failed */
    EWS_LOG_STATUS_AUTH_FORGERY,            /* bad message checksum */
    EWS_LOG_STATUS_AUTH_STALE,              /* authorization nonce stale */
    EWS_LOG_STATUS_AUTH_REQUIRED,           /* authorization not present */
    EWS_LOG_STATUS_AUTH_DIGEST_REQUIRED,    /* message digest not present */

    /*
     * 400 Bad Request
     */
    EWS_LOG_STATUS_BAD_REQUEST,             /* HTTP parse error */
    EWS_LOG_STATUS_BAD_FORM,                /* form data parse error */
    EWS_LOG_STATUS_BAD_IMAGEMAP,            /* imagemap query parse error */

    /*
     * Additional errors
     */
    EWS_LOG_STATUS_NOT_FOUND,               /* 404 not found or hidden */
    EWS_LOG_STATUS_METHOD_NOT_ALLOWED,      /* 405 method not allowed */
    EWS_LOG_STATUS_LENGTH_REQUIRED,         /* 411 length required */
    EWS_LOG_STATUS_UNAVAILABLE,             /* 503 aborted Document Fault */
    EWS_LOG_STATUS_NOT_IMPLEMENTED,         /* 501 bad method for URL */
    EWS_LOG_STATUS_NO_RESOURCES,            /* 500 insufficient resources */
    EWS_LOG_STATUS_INTERNAL_ERROR           /* 500 internal error */

  } EwsLogStatus;
```

```
/*
 * ewaLogHook
 * Application logging hook
 *
 * context - context of request being logged
 * status  - logging status
 */
#ifndef ewaLogHook
extern void ewaLogHook(EwsContext context, EwsLogStatus status);
#endif /* ewaLogHook */

#else /* EWA_LOG_HOOK */
#define ewaLogHook(context, status)
#endif /* EWA_LOG_HOOK */

/*
 * EmWeb/Server archive descriptor.
 */
typedef struct EwsArchive_s * EwsArchive;

/*
 * EmWeb/Server document handle
 */
typedef struct EwsDocument_s * EwsDocument;
#define EWS_DOCUMENT_NULL       ((EwsDocument) NULL)

/*
 * EmWeb/Server opaque read-only data handle
 */
typedef const uint8 EwsArchiveData[];

/*
 * EmWeb/Server authorization handle
 */
typedef struct EwsAuthHandle_s * EwsAuthHandle;
#define EWS_AUTH_HANDLE_NULL    ((EwsAuthHandle) NULL)

#ifdef EW_CONFIG_OPTION_FORM
/*
 * Application form interface constants for status byte
 */
#define EW_FORM_INITIALIZED     0x01
#define EW_FORM_DYNAMIC         0x02
#define EW_FORM_RETURNED        0x10
#define EW_FORM_PARSE_ERROR     0x20
#define EW_FORM_FILE_ERROR      0x20        /* equivalent to parse error */

/*
 * EmWeb/Server form element types
 */
#ifdef EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING
typedef struct EwsFormFieldHexString_s
  {
    uint32 length;
```

```
    uint8 *datap;
  } EwsFormFieldHexString, *EwsFormFieldHexStringP;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING */

#if (defined(EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC)       \
     || defined(EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC)    \
     || defined(EW_CONFIG_OPTION_FIELDTYPE_STD_MAC)     \
     )
typedef uint8 EwsFormFieldMAC[6];
#endif /* MAC */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_OID
typedef struct EwsFormFieldObjectID_s
  {
    uint32 length;      /* number of uint32's in datap */
    uint32 *datap;      /* array of uint32's representing OID */
  } EwsFormFieldObjectID, *EwsFormFieldObjectIDP;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_OID */

#if defined(EW_CONFIG_OPTION_FIELDTYPE_RADIO) ||        \
    defined(EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE)
/*
 * ewsFormEnumToString
 * Lookup string corresponding to enum from context
 *
 * context      - context of request
 * enum         - value of enumerator
 *
 * Returns string from corresponding HTML VALUE= field, or NULL if out of
 * bounds.
 */
extern const char * ewsFormEnumToString( EwsContext context, int value );
#endif /* EW_CONFIG_OPTION_FIELDTYPE_RADIO | SELECT_SINGLE */

#endif /* EW_CONFIG_OPTION_FORM */

/*
 * EwsRequestMethod
 *
 * This is an enum for supported request methods.  Bit values are used so
 * groups of valid request methods can be grouped into a single field
 */
typedef enum EwsRequestMethod_e
  {
    ewsRequestMethodUnknown = 0x0000    /* all others */
   ,ewsRequestMethodGet     = 0x0001    /* GET */
   ,ewsRequestMethodPost    = 0x0002    /* POST */
   ,ewsRequestMethodHead    = 0x0004    /* HEAD */

#  ifdef EW_CONFIG_OPTION_METHOD_OPTIONS
   ,ewsRequestMethodOptions = 0x0008    /* OPTIONS */
#  endif
#  ifdef EW_CONFIG_OPTION_METHOD_TRACE
   , wsRequestMethodTrace   = 0x0010    /* TRACE */
```

```
#  endif
#  ifdef EW_CONFIG_OPTION_METHOD_PUT
   , wsRequestMethodPut      = 0x0020     /* PUT */
#  endif
#  ifdef EW_CONFIG_OPTION_METHOD_DELETE
   ,ewsRequestMethodDelete   = 0x0040     /* DELETE */
#  endif

  } EwsRequestMethod;



#endif /* _EWS_DEF_H */
```

A-2.3. src/include/ews_sys.h

This file contains definitions and prototypes of the system functions of the application interface.

```
/*
 * ews_sys.h,v 1.17.2.3 1997/06/02 23:46:39 giusti Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server system interfaces
 *
 */

#ifndef _EWS_SYS_H
#define _EWS_SYS_H

#include "ew_types.h"
#include "ew_config.h"
#includ  " ws_def.h"
```

```
/*****************************************************************************
 *
 * INITIALIZATION AND SHUTDOWN
 *
 * The EmWeb/Server must be initialized at start-up before accepting requests,
 * installing document archives, or registering authentication information.
 *
 *****************************************************************************/

/*
 * ewsInit
 * Initialize EmWeb/Server
 *
 * Returns EWS_STATUS_OK on success, else failure code (TBD).
 */
extern EwsStatus ewsInit ( void );

#ifdef EW_CONFIG_OPTION_CLEANUP
/*
 * ewsShutdown
 * Graceful shutdown of EmWeb/Server terminating all requests in progress
 * and releasing all memory and buffers.
 *
 * Returns EWS_STATUS_OK on success, else failure code (TBD).
 */
extern EwsStatus ewsShutdown ( void );
#endif /* EW_CONFIG_OPTION_CLEANUP */

#ifdef EW_CONFIG_OPTION_SCHED
/*****************************************************************************
 *
 * SCHEDULING
 *
 * The EmWeb/Server is capable of both multi-threading HTTP requests and
 * yielding execution control to the CPU.  The following functions are provided
 * to give the application control over how EmWeb/Server schedules  pending
 * requests.
 *
 *****************************************************************************/

/*
 * ewsRun
 * Reschedule request processing after control returned to application as a
 * result of returning EWA_STATUS_OK_YIELD from ewaNetHTTPSend(),
 * ewaNetHTTPEnd(), or ewaDocumentFault(), or as the value to the status
 * parameter in ewsResume().
 *
 * Returns EWS_STATUS_OK on success, else failure code (TBD).
 */
extern EwsStatus ewsRun ( void );

#ifdef EW_CONFIG_OPTION_SCHED_SUSP_RES
/*
```

```
 * ewsSuspend
 * Suspend processing of the current request during an EmWeb callout function
 * (e.g. EmWebString, EmWebFormServe, or EmWebFormSubmit).  This function is
 * used in the implementation of proxies.
 *
 * Note that this does _not_ cause the server to stop processing requests; in
 * fact, it forces a flush of any partially filled buffer for the current
 * request and may process other pending requests.  This just causes the
 * current callout to be marked as 'suspended'.  When the current callout
 * returns, the result will be ignored - after ewsResume (below) is called
 * for this same context, the callout will be repeated.
 *
 * context      - context of request to be suspended
 *
 * Returns EWS_STATUS_OK on success, else failure code (TBD).
 */
extern EwsStatus ewsSuspend ( EwsContext context );

/*
 * ewsResume
 * Resume processing of a request previously suspended by ewsSuspend().  This
 * causes the request to be rescheduled, and the callout from which ewsSuspend
 * had been called will be reinvoked.
 *
 * context      - Context of suspended request to be resumed
 * status       - EWA_STATUS_OK or EWA_STATUS_OK_YIELD.
 *
 * Returns EWS_STATUS_OK on success, else failure code (TBD).
 */
EwsStatus ewsResume ( EwsContext context, EwaStatus status );
#endif /* EW_CONFIG_OPTION_SCHED_SUSP_RES */

#endif /* EW_CONFIG_OPTION_SCHED */

/*****************************************************************************
 *
 * MEMORY MANAGEMENT
 *
 * The EmWeb/Server presumes that the target O/S has a dynamic memory
 * management capability roughly equivalent to the POSIX malloc() and free()
 * functions.
 *
 * Alternatively, the application could create a static linked-list of memory
 * blocks of different sizes.  Refer to the product documentation for
 * information about run-time memory requirements.
 *
 * The application is responsible for providing the following memory management
 * functions to the EmWeb/Server.
 *
 *****************************************************************************/

#ifndef ewaAlloc
/*
 * ewaAlloc
```

```
 * Allocate a block of memory of at least the requested number of bytes.
 *
 * bytes        - requested size of the memory block in bytes
 *
 * Returns a pointer to the first byte of memory, or NULL if no memory of the
 * requested size is available.
 */
extern void * ewaAlloc ( uintf bytes );
#endif

#ifndef ewaFree
/*
 * ewaFree
 * Free a block of memory previously allocated by ewaAlloc()
 *
 * pointer      - pointer to beginning of memory block (from ewaAlloc())
 *
 * No return value
 */
extern void ewaFree ( void * pointer );
#endif

#ifdef EW_CONFIG_OPTION_DATE

/*****************************************************************************
 *
 * TIME-OF-DAY MANAGEMENT
 *
 * If DATE support is enabled, the EmWeb/Server invokes the following
 * application-provided function to retrieve the current date for use in
 * Date: HTTP headers.  Furthermore, the current date is also used in
 * Expire: and Last-modified: headers of dynamically generated documents
 * (i.e. HTML documents containing EMWEB extensions), if configured.
 * Note that the Last-modified: header of static documents use the archive
 * creation date stored in the archive by the EmWeb/Compiler.
 *
 *****************************************************************************/

#ifndef ewaDate
/*
 * ewaDate
 *
 * Return a string containing the current time-of-day in one of the HTTP
 * standard representations as follows:
 *
 * RFC1123: (Preferred standard)
 *   Fri, 28 Jun 1996 15:57:28 GMT
 *
 * RFC850: (Used, but not preferred)
 *   Friday, 28-Jun-96 15:57:28 GMT
 *
 * asctime: (Allowed, but discouraged)
 *   Jun  6 15:57:28 1996
 */
```

```
extern const char * ewaDate ( void );
#endif

#endif /* EW_CONFIG_OPTION_DATE */

#ifdef EW_CONFIG_OPTION_URL_HOOK

/*****************************************************************************
 *
 * URL REWRITING
 *
 * If URL rewriting support is enabled, the EmWeb/Server will invoke the
 * following function when each request has been successfully parsed, but
 * before the URL is looked up in the filesystem.  The application can then
 * return a new (or the same) URL to actually serve.
 *
 *****************************************************************************/

#ifndef ewaURLHook
/*
 * ewaURLHook
 *
 * context      - context of request
 * url          - requested URL
 *
 * Returns new URL, or NULL to cause abort.  (For no-op, simply return url).
 *
 * Note: the application MUST NOT invoke ewsNetHTTPAbort from here!  Return
 * a NULL pointer instead!
 *
 * The URL may be rewritten in place if the length is not increased;
 * if the returned pointer is not within the url parameter, the value is
 * copied by the server.
 */
extern char * ewaURLHook ( EwsContext context, char *url );
#endif

#endif /* EW_CONFIG_OPTION_LOG_HOOK */

/*
 * Since content length of zero is valid, we cannot use zero to
 * indicate when the content length is unknown (eg. dynamic data).
 * Assuming nobody will ever try to send/receive a 4gig document is
 * probably safe for now...
 */
#define EWS_CONTENT_LENGTH_UNKNOWN ((int32) -1)

#ifdef EW_CONFIG_OPTION_FILE
/*****************************************************************************
 *
 * LOCAL FILE SYSTEM SUPPORT
 *
 * The following routines must be supplied to support the server's
 * local file system.
```

```
 */

typedef union EwsFileParams_s
{
  /*
   * fileField - for support of the form INPUT TYPE=FILE field.
   * The server fills out this structure, and passes it to
   * the application when the file is submitted
   */
# ifdef EW_CONFIG_OPTION_FIELDTYPE_FILE
  struct
  {
    const char *fileName;         /* file name or NULL */
    const char *contentType;      /* MIME type */
    const char *contentEncoding;  /* content encoding or NULL */
    const char *contentDisposition; /* Content-disposition: */
    int32       contentLength;    /* length or EWS_CONTENT_LENGTH_UNKNOWN */
  } fileField;
# endif /* EW_CONFIG_OPTION_FIELDTYPE_FILE */


  /*
   * fileInfo - for support for local file operations (GET,HEAD,OPTIONS,
   * PUT,DELETE). This structure is setup by the application when a URL
   * that corresponds to a local file is received. This structure
   * gives the server all it needs to know to handle the file operation.
   */
# if defined( EW_CONFIG_OPTION_FILE_GET ) \
  || defined( EW_CONFIG_OPTION_FILE_PUT ) \
  || defined( EW_CONFIG_OPTION_FILE_DELETE )
  struct
  {
    EwaFileName fileName;         /* file name (opaque) */
    const char *contentType;      /* MIME type */
    const char *contentEncoding;  /* content encoding or NULL */
    const char *contentLanguage;  /* content language or NULL */
    const char *eTag;             /* reserved, MUST BE NULL */
    const char *lastModified;     /* modification time (RFC1123) or NULL */
    const char *lastModified1036;/* modification time (RFC1036) or NULL */
    const char *realm;            /* auth realm or NULL */
    int32       contentLength;    /* length or EWS_CONTENT_LENGTH_UNKNOWN */
    EwsRequestMethod   allow; /* reserved, MUST BE ewsRequestMethodGet */
    /* HEAD & OPTION _always_ allowed by default */

  } fileInfo;
# endif /* local file system GET/PUT/DELETE */

  char reserved1;                 /* prevent NULL union */

} EwsFileParams, *EwsFileParamsP;


/*
 *  waFileClose
```

```
 * Closes the file and indicates success or failure
 */
extern void
ewaFileClose ( EwaFileHandle handle, EwsStatus status );

/*
 * ewaFileWrite
 * Write data to a file
 */
extern sintf
ewaFileWrite ( EwsContext context
               , EwaFileHandle handle     /* handle from ewaFileOpen */
               , const uint8  *datap      /* pointer to data */
               , uintf         length     /* length of data */
               );
/* returns bytes written, <0 on error */

/*
 * ewaFileRead
 * Read data from a  file
 */
extern sintf
ewaFileRead ( EwsContext context
            , EwaFileHandle handle    /* handle from ewaFileOpen */
            , uint8 *datap           /* pointer to data buffer */
            , uintf length           /* length of buffer */
            );
/* returns bytes read, 0 on EOF, <0 on error */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_FILE
/*
 * ewaFilePost
 * For multipart/form-data file submissions.  Before the application's
 * submit function is called, the server may invoke this function for
 * each received file as part of a <INPUT TYPE=FILE> element in the
 * form.  If successful, the file handle will be passed to the
 * application's submit function at which point it is the application's
 * responsibility to close the file.
 */
extern EwaFileHandle
ewaFilePost ( EwsContext context
            ,const EwsFileParams *params
          );
/* returns EWA_FILE_HANDLE_NULL on error */
#endif /* EW_CONFIG_OPTION_FIELDTYPE_FILE */

#ifdef EW_CONFIG_OPTION_FILE_GET
/*
 * ewaFileGet
 * When the server determines that the file set by ewsContextSetFile
 * should be served to the browser, this function is invoked to open
 * the file for reading.
 *
 * To support Conditional GET, th  requ st message may include
```

```
 * If-Modified-Since, If-Unmodified-Since, If-Match,
 * If-None-Match, or If-Range headers.  In this case, the
 * server will use the information given in the returned
 * params block to determine if the file should be retrieved.
 */
extern EwaFileHandle
ewaFileGet ( EwsContext context
           ,const char *url
           ,const EwsFileParams *params /* from ewsContextSetFile() */
           );
/* returns EWA_FILE_HANDLE_NULL on error */
#endif /* EW_CONFIG_OPTION_FILE_GET */

#endif /* EW_CONFIG_OPTION_FILE */

#endif /* _EWS_SYS_H */
```

A-2.4. src/include/ews_net.h

This file contains the definitions and prototypes used by the networking functions of the application interface.

```
/*
 * ews_net.h,v 1.20.2.1 1997/06/15 03:03:18 giusti Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server application interface to network transport layer
 *
 */
#ifndef _EWS_NET_H
#define _EWS_NET_H

#include "ew_types.h"
#include "ew_config.h"
```

```
#include "ews_def.h"

/*****************************************************************************
 *
 * In a typical TCP/IP implementation, the application is responsible for
 * listening to the HTTP TCP port (80) for connection requests.  When a
 * connection request is received, the application accepts the connection
 * on behalf of EmWeb/Server and invoke ewsNetHTTPStart() to inform the
 * EmWeb/Server of the new request.
 *
 * EmWeb/Server assumes that the application maintains data buffers for
 * the reception and transmission of TCP data.  The only requirements that
 * EmWeb/Server imposes on the buffer implementation is as follows:
 *
 *   1. Buffers can be uniquely identified by a buffer descriptor.  No
 *      assumptions are made about the actual structure of the buffer
 *      descriptors or their relationship to data.  For example, a buffer
 *      descriptor could be an index into a table, a pointer to a
 *      structure (either contiguous or seperate from the data represented),
 *      etc.  The application is responsible for defining the appropriate
 *      type for EwaNetBuffer and value for EWA_NET_BUFFER_NULL.
 *
 *   2. Buffers can be chained together.  Given a buffer descriptor,
 *      EmWeb/Server must be able to get or set the "next buffer in the chain"
 *      field.  This is done by ewaNetBufferNextSet() and
 *      ewaNetBufferNextGet().  Note that the buffer chain is terminated when
 *      the next buffer value is EWA_NET_BUFFER_NULL.
 *
 *   3. Given a buffer descriptor, EmWeb/Server can determine the start of
 *      data in the buffer.  Additionally, EmWeb/Server may change the start
 *      of data in the buffer (EmWeb/Server only changes the start of data in
 *      the buffer upward).  This is done by ewaNetBufferDataGet() and
 *      ewaNetBufferDataSet().
 *
 *   4. Given a buffer descriptor, EmWeb/Server can determine the size of
 *      contiguous data in the buffer.  Additionally, EmWeb/Server may
 *      change the size of the buffer (EmWeb/Server only changes the
 *      size of the buffer downward).  This is done by ewaNetBufferLengthGet()
 *      and ewaNetBufferLengthSet().
 *
 *   5. EmWeb/Server may allocate a buffer by invoking ewaNetBufferAlloc().
 *      If no buffers are available, this function returns EWA_NET_BUFFER_NULL.
 *      Additionally, EmWeb/Server may release a buffer by invoking
 *      ewaNetBufferFree().
 *
 * As the application receives TCP data on an HTTP connection, it passes
 * this data to the EmWeb/Server by invoking ewsNetHTTPReceive().
 *
 * The EmWeb/Server transmits TCP data on an HTTP connection by invoking
 * ewaNetHTTPSend().  The application may throttle EmWeb/Server by
 * returning EWA_STATUS_OK_YIELD.  This causes the EmWeb/Server to
 * save state and return control to the application.  The application must
 * invoke ewsRun() to give the EmWeb/Server an opportunity to continue
 * proc ssing the r qu st.
```

```
 *
 * When the EmWeb/Server completes a request, it invokes ewaNetHTTPEnd().
 *
 * The application may abort a request at any time by invoking
 * ewsNetHTTPAbort().
 *
 ************************************************************************/

/*
 * ewsNetHTTPStart
 * Start a new HTTP request
 *
 * net_handle    - application-specific handle representing request
 *
 * Returns context for the request, or EWS_CONTEXT_NULL on failure.
 */
extern EwsContext ewsNetHTTPStart ( EwaNetHandle net_handle );

/*
 * ewsNetHTTPAbort
 * Abort a previously started HTTP request.
 *
 * context       - context of request to be aborted
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD)
 */
extern EwsStatus ewsNetHTTPAbort ( EwsContext context );

/*
 * ewsNetHTTPReceive
 * Receive request data from the network.
 *
 * context       - context of request to which received data applies
 * buffer        - buffer containing received data
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD)
 */
extern EwsStatus ewsNetHTTPReceive ( EwsContext context, EwaNetBuffer buffer );

#ifdef EW_CONFIG_OPTION_SCHED_FC

/*
 * ewsNetFlowControl
 * Mark context for flow control to avoid predicted congestion.  ewsRun() will
 * place the context on the suspended list at the next opportunity and
 * continue by processing additional requests.
 *
 * context       - context of request to be flow controlled
 *
 * Returns EWS_STATUS_OK.
 */
EwsStatus ewsNetFlowControl ( EwsContext context ) ;

/*
```

```
 * ewsNetUnFlowControl
 * Resume previously flow controlled context
 *
 * context      - context of request to be flow controlled
 *
 * Returns EWS_STATUS_OK.
 */
EwsStatus ewsNetUnFlowControl ( EwsContext context );

#endif /* EW_CONFIG_OPTION_SCHED_FC */

#ifndef ewaNetHTTPSend
/*
 * ewaNetHTTPSend (Application)
 * This function must be provided by the application to accept data from
 * EmWeb/Server for transmission to a browser in response to a request.
 *
 * net_handle   - application-specific request handle from ewsNetHTTPStart()
 * buffer       - buffer containing data to be transmitted
 *
 * Returns EWA_STATUS_OK on success, EWA_STATUS_OK_YIELD on success and
 * request EmWeb/Server to yield CPU to application, or EWA_STATUS_ERROR on
 * failure.
 */
extern EwaStatus ewaNetHTTPSend
  ( EwaNetHandle net_handle, EwaNetBuffer buffer );
#endif

#ifndef ewaNetHTTPEnd
/*
 * ewaNetHTTPEnd (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to indicate the completion of a request after all response
 * data has been sent.
 *
 * net_handle   - application-specific request handle from ewsNetHTTPStart()
 *
 * Returns EWA_STATUS_OK on success, EWA_STATUS_OK_YIELD on success and request
 * EmWeb/Server to yield CPU to application, or EWA_STATUS_ERROR on failure.
 */
extern EwaStatus ewaNetHTTPEnd ( EwaNetHandle net_handle );
#endif

#ifndef ewaNetBufferAlloc
/*
 * ewaNetBufferAlloc (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to request a network buffer to be used for sending data.
 *
 * Returns a buffer, or EWA_NET_BUFFER_NULL on failure. The length of the
 * buffer must be initialized to the number of bytes available for use
 * by EmWeb/Server.
 */
extern EwaNetBuffer  waN tBufferAlloc ( void );
```

```
#endif

#ifndef ewaNetBufferFree
/*
 * ewaNetBufferFree (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to release one or a chain of network buffers (either from
 * ewsNetHTTPReceive() or ewaNetBufferAlloc()).
 *
 * buffer       - buffer(s) to be released
 *
 * No return value
 */
extern void ewaNetBufferFree ( EwaNetBuffer buffer );
#endif

#ifndef ewaNetBufferLengthGet
/*
 * ewaNetBufferLengthGet (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to get the length of the data portion of this buffer fragment.
 *
 * buffer       - buffer descriptor
 *
 * Returns length of buffer
 */
uintf ewaNetBufferLengthGet ( EwaNetBuffer buffer );
#endif

#ifndef ewaNetBufferLengthSet
/*
 * ewaNetBufferLengthSet (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to set the length of the data portion of this buffer fragment.
 * This function is only used to decrease the original length of a
 * buffer.  EmWeb/Server never increases the length of a buffer.
 *
 * buffer       - buffer descriptor
 * length       - new length value
 *
 * No return value
 */
void ewaNetBufferLengthSet ( EwaNetBuffer buffer, uintf length );
#endif

#ifndef ewaNetBufferDataGet
/*
 * ewaNetBufferDataGet (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to get the pointer to the data contained in the buffer.
 *
 * buffer       - buffer descriptor
 *
 * Returns pointer to data in buffer
```

```
 */
uint8 * ewaNetBufferDataGet ( EwaNetBuffer buffer );
#endif

#ifndef ewaNetBufferDataSet
/*
 * ewaNetBufferDataSet (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to set the pointer to the data contained in the buffer.
 * This function is only used to advance the start of data forward.
 * EmWeb/Server never moves this pointer backward.
 *
 * buffer       - buffer descriptor
 * datap        - new start of data value
 *
 * No return value
 */
void ewaNetBufferDataSet ( EwaNetBuffer buffer, uint8 *datap );
#endif

#ifndef ewaNetBufferNextGet
/*
 * ewaNetBufferNextGet (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to get the next buffer descriptor in the buffer chain.
 *
 * buffer       - buffer descriptor
 *
 * Returns next buffer descriptor, or EWA_NET_BUFFER_NULL if end of chain.
 */
EwaNetBuffer ewaNetBufferNextGet ( EwaNetBuffer buffer );
#endif

#ifndef ewaNetBufferNextSet
/*
 * ewaNetBufferNextSet (Application)
 * This function must be provided by the application.  It is invoked by
 * EmWeb/Server to set the next buffer descriptor in the buffer chain.
 *
 * buffer       - buffer descriptor
 * next         - next buffer descriptor to be attached to buffer
 *
 * No return value
 */
void ewaNetBufferNextSet ( EwaNetBuffer buffer, EwaNetBuffer next );
#endif

#ifndef ewaNetLocalHostName
/*
 * ewaNetLocalHostName (Application)
 * This function must be provided by the application.  It is invoked by the
 * EmWeb/Server to build proper redirection (Location:) headers.  This can
 * be either a dotted IP address or a fully qualified hostname.
 */
```

```
const char * ewaNetLocalHostName ( EwsContext context );
#endif

#ifndef ewaNetHTTPCleanup
/*
 * ewaNetHTTPCleanup
 * This function must be provided by the application or defined as a
 * empty macro.  It is invoked by EmWeb/Server when a request completes,
 * allowing the application to reset any processing state stored in the
 * network handle.  Note that for HTTP 1.1 persistent connections, this
 * routine may be called several times for the same connection, as one
 * connection can be used for multiple requests.  For the last request
 * on a connection, this routine will be called before ewaNetHTTPEnd is
 * invoked.
 *
 * Returns void.  Note that the request context is undefined during
 * this call and cannot be accessed.
 */
void ewaNetHTTPCleanup( EwaNetHandle handle );
#endif
#endif /* _EWS_NET_H */
```

A-2.5. src/include/ews_doc.h

This file contains the definitions and prototypes for the archive and document management functions of the application interface.

```
/*
 * ews_doc.h,v 1.20 1997/05/21 21:55:44 ian Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server application interface to document archives
 *
 */
```

```
#ifndef _EWS_DOC_H
#define _EWS_DOC_H

#include "ew_types.h"
#include "ew_config.h"
#include "ews_def.h"


/*****************************************************************************
 * ARCHIVE MAINTENANCE
 *
 * The EmWeb/Compiler generates an archive of one or more documents.  Documents
 * can be HTML files, JAVA programs, graphical images, or any other information
 * resource addressable by a URL.  Archives may be independently loaded or
 * unloaded into the EmWeb/Server.
 *
 * In most applications, the entire set of available documents would be
 * compiled into a single archive and loaded at boot time.  However, some
 * applications may desire to dynamically load archives at run-time as needed
 * in order to reduce memory requirements.  In fact, some applications may
 * want to implement a scheme similar to page swapping under some operating
 * systems to cache an active set of documents in memory while storing other
 * documents in a secondary storage area.  Such a secondary storage area
 * could be in FLASH memory, or on a remote server using TFTP or other
 * protocols to load documents at run-time.
 *
 * An EmWeb archive consists of two components.  First, there is the archive
 * data component containing the database of compressed documents, information
 * about how to construct dynamic documents at run-time, access controls,
 * etc.  Second, there is the archive object component containing the run-time
 * object code used for the construction of dynamic documents, etc.
 *
 * Operating systems supporting the run-time loading and linking of object
 * code may offload both the data and object archive components to a secondary
 * storage area.  Otherwise, only the data components would be offloaded
 * while the object components would be statically linked into the run-time
 * executable image.
 *
 * Each archive contains an archive descriptor in the object component.  The
 * archive descriptor is referenced by a public symbol generated by the
 * EmWeb/Compiler.  In order to activate an archive at run-time, the
 * application must invoke ewsDocumentInstallArchive() with parameters
 * indicating the address of the data component and the archive descriptor of
 * the object component.  The archive may be deactivated by invoking
 * ewsDocumentRemoveArchive().
 *
 *****************************************************************************/

/*
 * ewsDocumentInstallArchive
 *
 * Install a document archive g nerated by EmWeb/Compiler into EmWeb/Server
 * run-time database.
 *
```

```
 * descriptor    - public symbol of archive object component descriptor
 * datap         - pointer to archive data component
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD)
 */
extern EwsStatus ewsDocumentInstallArchive
  ( EwsArchive descriptor, EwsArchiveData datap );

#ifdef EW_CONFIG_OPTION_CLEANUP

/*
 * ewsDocumentRemoveArchive
 *
 * Removes previously installed document archive from run-time database.
 *
 * descriptor    - public symbol of archive object component descriptor
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD)
 */
extern EwsStatus ewsDocumentRemoveArchive ( EwsArchive descriptor );

#endif /* EW_CONFIG_OPTION_CLEANUP */

/*
 * ewsDocumentArchiveName
 *
 * Returns the archive name as stored in the archive by the EmWeb/Compiler.
 * or NULL on error.
 */
extern const char * ewsDocumentArchiveName ( const uint8 *datap );

/*
 * ewsDocumentArchiveDate
 *
 * Returns the RFC1123 date string stored in the archive header generated
 * by the EmWeb/Compiler when the archive was created, or NULL on error.
 *
 * There is also a routine for the less preferred RFC1036 representation.
 */
extern const char * ewsDocumentArchiveDate ( const uint8 *datap );
extern const char * ewsDocumentArchiveDate1036 ( const uint8 *datap );

#ifdef EW_CONFIG_OPTION_DEMAND_LOADING

/*****************************************************************************
 *
 * DEMAND LOADING
 *
 * In order to implement on-demand archive loading, the application may
 * register document URLs with the EmWeb/Serv r which are valid but not
 * l aded.  This is don  by invoking  wsDocumentRegister().  If a r gistered
 * d cument is requested, the EmWeb/Server will notify the application by
 * invoking ewaDocum ntFault().  At this point, the application may load
 * a new archive (possibly removing a previously installed archive to make
```

```
 * room).  When the archive containing the page is installed, the EmWeb/Server
 * will automatically complete processing the request.  The request can be
 * aborted either immediately by returning EWA_STATUS_ERROR from the
 * ewaDocumentFault() function, or by invoking ewsNetHTTPAbort().
 *
 * Once a document is registered, there is no need to re-register it in the
 * event that the corresponding archive is removed.  EmWeb/Server remembers
 * that the document has been registered as dynamically loadable.  However,
 * the application may deregister a document by invoking
 * ewsDocumentDeregister().
 *
 ************************************************************************/

/*
 * ewsDocumentRegister
 * Registers the URL of a document with the run-time database indicating that
 * the given document is valid but not currently loaded into memory.
 *
 * url          - local URL of document to be registered
 * handle       - application-specific handle passed to ewaDocumentFault()
 *
 * Returns document descriptor, or EWS_DOCUMENT_NULL on failure.
 */
extern EwsDocument ewsDocumentRegister
  ( const char * url, EwaDocumentHandle handle );

/*
 * ewsDocumentDeregister
 * Unregisters a previously registered document from the run-time database
 * indicating that the given dynamically-loadable document is no longer valid.
 *
 * document     - descriptor of document to be deregistered
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD).
 */
extern EwsStatus ewsDocumentDeregister ( EwsDocument document );

#ifndef ewaDocumentFault
/*
 * ewaDocumentFault (Application)
 * This function must be provided by the application, and is invoked by
 * EmWeb/Server when a registered document has been requested but is not
 * currently loaded into memory.
 *
 * context      - context of request for document
 * handle       - application-specific handle from ewsDocumentRegister()
 *
 * Returns EWA_STATUS_OK on succ ss, EWA_STATUS_OK_YIELD on success with
 * requ st for EmWeb/Server to yield the CPU, or EWA_STATUS_ERROR on failure.
 * If failure status is returned, the r qu st is aborted.  Otherwise, the
 * request will complet  after the document has been loaded into memory.
 */
extern EwaStatus ewaDocumentFault
  ( EwsContext context, EwaDocumentHandle handle );
```

```
#endif

#endif /* EW_CONFIG_OPTION_DEMAND_LOADING */

#ifdef EW_CONFIG_OPTION_CLONING

/*****************************************************************************
 *
 * DOCUMENT CLONING
 *
 * Documents may be cloned and assigned to a new URL.  This allows multiple
 * instances of a document to exist.  An application-specific handle can be
 * used to identify an instance of a document from the request context.
 *
 * The application clones a document by invoking ewsDocumentClone().  The
 * cloned document may be removed by invoking ewsDocumentRemove().  All
 * clones created from documents in an archive must be removed before the
 * archive can be removed.
 *
 *****************************************************************************/

/*
 * ewsDocumentClone
 * Clone an existing document under a new URL in the database.
 *
 * baseurl      - URL of existing document
 * newurl       - URL of new cloned document
 * handle       - application-specific handle available in request context
 *
 * Returns document descriptor or EWS_DOCUMENT_NULL on error.
 */
extern EwsDocument ewsDocumentClone
  ( const char * baseurl, const char * newurl, EwaDocumentHandle handle );

/*
 * ewsDocumentRemove
 * Remove a previously cloned document from the database.
 *
 * document     - document descriptor of previously cloned document.
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD).
 */
extern EwsStatus ewsDocumentRemove ( EwsDocument document );
#endif /* EW_CONFIG_OPTION_CLONING */

#ifdef EW_CONFIG_OPTION_DOCUMENT_DATA
/*
 * ewsDocumentData
 * Retrieve pointer to raw data and data size from URL in archive
 *
 * url          - url of document
 * bytesp       - output: size of document (as-is in archive) in bytes
 * datapp       - output: pointer to start of document
 *
```

```
 * Returns EWS_STATUS_OK on success, else error code (TBD).
 */
extern EwsStatus ewsDocumentData ( const char * url
                                  ,uint32 *bytesp
                                  ,const uint8 **datapp );
#endif /* EW_CONFIG_OPTION_DOCUMENT_DATA */

#ifdef EW_CONFIG_OPTION_DOCUMENT_SET_REALM
#ifdef EW_CONFIG_OPTION_AUTH
/*
 * ewsDocumentSetRealm
 * Set or change authentication realm of document at run-time.
 *
 * url          - url of document
 * realm        - name of authentication realm, or NULL (or "") to disable.
 *
 * Returns EWS_STATUS_OK on success, else error code.
 */
extern EwsStatus ewsDocumentSetRealm ( const char * url, const char *realm );
#endif /* EW_CONFIG_OPTION_AUTH */
#endif /* EW_CONFIG_OPTION_DOCUMENT_SET_REALM */

#endif /* _EWS_DOC_H */
```

A-2.6. src/include/ews_auth.h

This file contains the definitions and prototypes of the authorization and security functions of the application interface.

```
/*
 * ews_auth.h,v 1.18 1997/05/21 21:55:44 ian Exp
 *
 * Product:     EmWeb
 * Release:     R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server application interface to authorization database
```

```
 *
 */

#ifndef _EWS_AUTH_H_
#define _EWS_AUTH_H_

#ifdef EW_CONFIG_OPTION_AUTH

#include "ew_types.h"
#include "ew_config.h"

/*****************************************************************************
 *
 * HTTP Authorization techniques are discussed in the HTTP specifications.
 * Knowledge of these principles are essential for correct use of these
 * authorization interfaces.
 *
 * One or more HTML documents can be assocated with a single "realm".  A realm
 * is a case-sensitive ascii string that defines a "protection space" for the
 * associated document(s).
 *
 * Each realm can have zero or more authentication entries associated with it.
 * If a realm has no authentication entries, then it is considered
 * "unprotected" by the server, and its acess will always be granted to any
 * requesting client.
 *
 * For those realms with authentication entries, any client that attempts to
 * access any document associated with that realm will be required to
 * authenticate itself.  To authenticate, the client must send authentication
 * parameters that validate against at least one authentication entry for that
 * document's realm.  Clients that do not authenticate are not served the
 * requested document.
 *
 * For example, assume a realm exists called "foo".  It has three
 * authentication entries associated with it (2 using the "basic" scheme
 * defined in the HTTP specification, and one other ficticious type):
 *
 * REALM: "foo"
 *         Authentication Entry 1:
 *              Type: "basic"
 *              parameters: Username="user1"
 *                          Password="guest"
 *              EwaAuthHandle:  <application's entry1 identifier>
 *
 *         Authentication Entry 2:
 *              Type: "future"
 *              parameters: index=37
 *                          cookie="837I8U9"
 *                          token="S37"
 *              EwaAuthHandle:  <application's entry2 identifier>
 *
 *         Authentication Entry 3:
 *              Type: "basic"
 *              parameters: Username="sinclair"
```

```
 *                    Password="babylon"
 *              EwaAuthHandle:  <application's entry3 identifier>
 *
 * When a client attempts to access a document associated with realm "foo", it
 * will need to authenticate against one of the above entries.  Which one the
 * client authenticates against is at its disgression.
 *
 * When a client does authenticate against one of the above entries, the
 * EwaAuthHandle for that entry is stored in the current context for the
 * document (EwsContext).  The datatype of this EwaAuthHandle is
 * implementation-defined.
 ************************************************************************/

/* ------------------------------------------------------------
 * Types & Constants
 * ------------------------------------------------------------ */

/*
 * Defines the authorization schemes supported by the EmWeb server.
 * New schemes will be added as they are supported.
 */
typedef enum
{

#   ifdef EW_CONFIG_OPTION_AUTH_BASIC
    ewsAuthSchemeBasic,
#   endif /* EW_CONFIG_OPTION_AUTH_BASIC */

#   ifdef EW_CONFIG_OPTION_AUTH_DIGEST
    ewsAuthSchemeDigest,
#   endif /* EW_CONFIG_OPTION_AUTH_DIGEST */

#   ifdef EW_CONFIG_OPTION_AUTH_MBASIC
    ewsAuthSchemeManualBasic,
#   endif /* EW_CONFIG_OPTION_AUTH_MBASIC */

#   ifdef EW_CONFIG_OPTION_AUTH_MDIGEST
    ewsAuthSchemeManualDigest,
#   endif /* EW_CONFIG_OPTION_AUTH_MDIGEST */

    ewsAuthMaxScheme            /* count of supported schemes */

} EwsAuthScheme;


/*
 * this structure defines parameters for all the supported
 * authorization types.  New parameters will be added in
 * to this structure in the future
 */
typedef union
{
#ifdef EW_CONFIG_OPTION_AUTH_BASIC
  struct
```

```
  {
        char  *userName;
        char  *passWord;
  } basic;
#endif /* EW_CONFIG_OPTION_AUTH_BASIC */
#if defined(EW_CONFIG_OPTION_AUTH_DIGEST) ||
defined(EW_CONFIG_OPTION_AUTH_MDIGEST)
  struct
  {
        char    *userName;
        char    *passWord;

#       ifdef EW_CONFIG_OPTION_AUTH_DIGEST_M
        boolean digestRequired; /* require client message data verification */
#       endif /* EW_CONFIG_OPTION_AUTH_DIGEST_M */

  } digest;
#endif /* EW_CONFIG_OPTION_AUTH_DIGEST || EW_CONFIG_OPTION_AUTH_MDIGEST */
  char reserved1;
} EwsAuthParameters;


/*
 * This structure represents a single authorization entry.
 */
typedef struct
{
  EwsAuthScheme     scheme;
  EwsAuthParameters params;
  EwaAuthHandle     handle;     /* user defined */
} EwsAuthorization;


/* ------------------------------------------------------------
 * Interfaces
 * ------------------------------------------------------------ */


/*
 * ewsAuthRegister
 * And a new authorization entry to the realm.
 *
 * realm - Input, case sensitive asciiz realm name.
 * authorization - Input, buffer containing authorization entry information.
 *                 This buffer is free for use by the caller on return.
 *
 * Returns a non-NULL handle to the authorization entry, or NULL on failure.
 */

extern EwsAuthHandle ewsAuthRegister
  ( const char *realm,const EwsAuthorization *authorization );


/*
```

```
 * ewsAuthRemove
 * Deletes a particular authentication entry identified by handle.
 *
 * handle - Input, The handle that was returned by ewsAuthRegister() for
 *          this entry.
 *
 * Returns EWS_STATUS_OK on success, else failure code.
 */
extern EwsStatus ewsAuthRemove ( EwsAuthHandle handle );

/*
 * ewsAuthRealmEnable
 * Enables a realm to restrict access to documents (restricted by default)
 *
 * realm - Input, the realm to restrict
 *
 * Returns EWS_STATUS_OK on success, else failure code
 */
extern EwsStatus ewsAuthRealmEnable ( const char * realm );

/*
 * ewsAuthRealmDisable
 * Disables a realm causing documents in that realm to be generally accessible.
 *
 * realm - Input, the realm to restrict
 *
 * Returns EWS_STATUS_OK on success, else failure code
 */
extern EwsStatus ewsAuthRealmDisable ( const char * realm );

/*
 * ewaAuthRealmQualifier
 * Returns realm qualifier string to append to realm names, or NULL
 */
#ifndef ewaAuthRealmQualifier
extern const char *ewaAuthRealmQualifier ( void );
#endif /* ewaAuthRealmQualifier */

#if defined(EW_CONFIG_OPTION_AUTH_DIGEST) \
 || defined(EW_CONFIG_OPTION_AUTH_MDIGEST)
/*
 * ewaAuthNonceCreate
 * Create raw input used to generate one-time authentication challenges
 * given a request context and realm.
 *
 * context - request context
 * realm   - null-terminated string containing name of realm
 * noncep  - pointer to application-specific nonce parameters
 *
 * No return value
 */
extern void ewaAuthNonceCreate
  ( EwsContext context, const char * realm, EwaAuthNonce *noncep );
```

```
/*
 * ewaAuthNonceCheck
 * Validate that the previously created nonce is valid given the current
 * request.
 *
 * context - request context
 * realm   - null-terminated string containing name of realm
 * noncep  - pointer to requested nonce parameters
 * count   - previous use count (0, 1, ...) of nonce value
 *
 * Returns ewaAuthNonceOK on success, ewaAuthNonceStale if valid but expired,
 * or ewaAuthNonceDenied if nonce is not permitted given the request context.
 */
typedef enum EwaAuthNonceStatus_e
  {
    ewaAuthNonceOK,      /* nonce value valid for request */
    ewaAuthNonceLastOK,  /* nonce value valid, but won't be again */
    ewaAuthNonceStale,   /* nonce value is stale, generate new nonce */
    ewaAuthNonceDenied   /* nonce value is invalid */
  } EwaAuthNonceStatus;

extern EwaAuthNonceStatus ewaAuthNonceCheck
  ( EwsContext context, const char * realm, EwaAuthNonce *noncep, uintf count );
#endif /* EW_CONFIG_OPTION_AUTH_(M)DIGEST */


#ifdef EW_CONFIG_OPTION_AUTH_MBASIC
/*
 * ewaAuthCheckBasic
 * Given the current context, realm, base64 cookie, and optionally
 * decoded username/password text, determine if the authorization should
 * be granted or denied.
 *
 * context - (input) request context
 * realm   - (input) null-terminated string containing name of realm
 * basicCookie - (input) the base64 encoding of the text user:password,
 *           as described in the HTTP RFC.
 * userName - (input) if EW_CONFIG_OPTION_AUTH_MBASIC_DECODE is defined,
 *           then this parameter points to a null terminated string
 *           containing the user name decoded from the basicCookie.  If
 *           EW_CONFIG_OPTION_AUTH_MBASIC_DECODE is not defined, this
 *           is a NULL pointer.
 * password - (input) if EW_CONFIG_OPTION_AUTH_MBASIC_DECODE is defined,
 *           then this parameter points to a null terminated string
 *           containing the password decoded from the basicCookie.  If
 *           EW_CONFIG_OPTION_AUTH_MBASIC_DECODE is not defined, this
 *           is a NULL pointer.
 *
 * R turn value: boolean, if TRUE, then the authorization is granted and
 * the document is served.  Otherwise, authorization is denied.
 *
 * This function can be supended by ewsSuspend().  It's return values
 * are ignored, and it will be recalled with the same parameters when
 * the context is resumed using ewsResume().
```

```
 */

#ifndef ewaAuthCheckBasic
extern boolean ewaAuthCheckBasic( EwsContext context
                                 ,const char *realm
                                 ,const char *basicCookie
                                 ,const char *userName
                                 ,const char *password );
#endif /* ewaAuthCheckBasic */
#endif /* EW_CONFIG_OPTION_AUTH_MBASIC */


#ifdef EW_CONFIG_OPTION_AUTH_MDIGEST
/*
 * ewaAuthCheckDigest
 * Given the current context, realm, username, and nonce,
 * return the corresponding MD5 hash of the username:realm:password
 * text.  This text will be used by the server to verify the
 * authorization attempt by the client.
 *
 * context - (input) request context
 * realm   - (input) null-terminated string containing name of realm
 *           taken from the request header.
 * nonce   - (input) null-terminated string containing the nonce that
 *           was supplied to the server.
 * userName - (input) this parameter points to a null terminated string
 *           containing the user name as supplied in the authorization
 *           request header.
 * digest  - (output) on return, *digest will point to a null-terminated
 *           ascii text string representing the MD5 checksum of the
 *           username, realm and password (MD5( username:realm:password))
 *           as described in the Digest Access Authentication RFC2069
 *
 * Return value: boolean, if TRUE, then *digest points to a valid MD5 hash of
 * username:realm:password, which will be used by the server to complete
 * the authorization process (which may or may not be granted).  If FALSE,
 * the authorization is immediately denied.
 *
 * This function can be supended by ewsSuspend().  It's return values
 * are ignored, and it will be recalled with the same parameters when
 * the context is resumed using ewsResume().
 */

#ifndef ewaAuthCheckDigest
extern boolean ewaAuthCheckDigest( EwsContext context
                                  ,const char *realm
                                  ,const EwaAuthNonce *noncep
                                  ,const char *userName
                                  ,const char **digest );
# ndif /* ewaAuthCh ckDigest */
#endif /* EW_CONFIG_OPTION_AUTH_MDIGEST */


#ifdef EW_CONFIG_OPTION_AUTH_VERIFY
```

```
/*
 * ewaAuthVerifySecurity
 * Called _after_ the server has determined that the authorization
 * will be granted for the request, this function allows the application
 * to "short circuit" the authorization based on information about
 * the request (eg, IP address of client, etc.).  The method used
 * to determine whether or not the request is authorized is proprietary
 * to the application.
 *
 * context - (input) request context
 * realm   - (input) null-terminated string containing name of realm
 *           taken from the request header.
 *
 * Return value: boolean, if TRUE, then the authorization is granted, and
 * the document will be served to the client.  If FALSE,
 * the authorization is immediately denied.
 *
 * This function can be supended by ewsSuspend().  It's return values
 * are ignored, and it will be recalled with the same parameters when
 * the context is resumed using ewsResume().
 */

#ifndef ewaAuthVerifySecurity
extern boolean ewaAuthVerifySecurity( EwsContext context
                                     ,const char *realm );
#endif /* ewaAuthVerifySecurity */
#endif /* EW_CONFIG_OPTION_AUTH_VERIFY */


/*
 *** Local Variables: ***
 *** mode: c ***
 *** tab-width: 4 ***
 *** End: ***
 */

#endif /* EW_CONFIG_OPTION_AUTH */

#endif /* _EWS_AUTH_H_ */
```

A-2.7. src/include/ews_ctxt.h

This file contains definitions and prototypes of the request context access functions of the application interface.

```
/*
 * ews_ctxt.h,v 1.14 1997/05/25 23:30:32 giusti Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
```

```
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sales@agranat.com, support@agranat.com
 *
 * http://www.agranat.com/
 *
 * EmWeb/Server public definitions
 *
 */
#ifndef _EWS_CONTEXT_H
#define _EWS_CONTEXT_H

#include "ew_types.h"
#include "ew_config.h"
#include "ews_def.h"

/*
 * REQUEST CONTEXT
 *
 * Each HTTP request received by EmWeb/Server is assigned a unique context.
 * The context structure contains all the information pertaining to the
 * request, as well as internal state information regarding the processing
 * state of the request.
 *
 * API functions are provided to give the application some visibility into
 * the current context state.
 */

/*
 * ewsContextNetHandle
 * Return the application-specific network handle provided to EmWeb/Server
 * by the application in ewsNetHTTPStart().
 */
EwaNetHandle ewsContextNetHandle ( EwsContext context );

#if defined (EW_CONFIG_OPTION_DEMAND_LOADING) \
    || defined (EW_CONFIG_OPTION_CLONING)
/*
 * ewsContextDocumentHandle
 * Return the application-specific document handle provided to EmWeb/Server
 * by the application in ewsDocumentRegist r() or  wsDocumentClone().
 */
EwaDocumentHandle ewsCont xtDocumentHandle ( EwsContext context );
#endif /* EW_CONFIG_OPTION_DEMAND_LOADING || EW_CONFIG_OPTION_CLONING */
```

```
#ifdef EW_CONFIG_OPTION_AUTH
/*
 * ewsContextAuthHandle
 * Return the application-specific authorization handle provided to
 * EmWeb/Server by the application in ewsAuthRegister().
 */
EwaAuthHandle ewsContextAuthHandle ( EwsContext context );
#endif /* EW_CONFIG_OPTION_AUTH */

#ifdef EW_CONFIG_OPTION_SCHED_SUSP_RES
/*
 * ewsContextIsResuming
 * Return TRUE if EmWeb/Server is resuming (as a result of ewsResume()) after
 * the context was suspended (as a result of ewsSuspend()).
 */
boolean ewsContextIsResuming ( EwsContext context );
#endif /* EW_CONFIG_OPTION_SCHED_SUSP_RES */

#if defined (EW_CONFIG_OPTION_STRING) || defined (EW_CONFIG_OPTION_INCLUDE)
/*
 * ewsContextIterations
 * Returns the iteration count corresponding for EMWEB_ITERATE
 */
uint32 ewsContextIterations ( EwsContext context );
#endif /* EW_CONFIG_OPTION_STRING || EW_CONFIG_OPTION_INCLUDE */

#ifdef EW_CONFIG_OPTION_CONTEXT_SEND_REPLY
/*
 * ewsContextSendReply
 * Serves specified local path URL to browser in response to form submission
 * or raw CGI.
 */
EwsStatus ewsContextSendReply ( EwsContext context, char * url );
#endif /* EW_CONFIG_OPTION_CONTEXT_SEND_REPLY */

#if defined( EW_CONFIG_OPTION_FILE_GET ) \
 || defined( EW_CONFIG_OPTION_FILE_PUT ) \
 || defined( EW_CONFIG_OPTION_FILE_DELETE )
EwsStatus ewsContextSetFile( EwsContext context
                            ,EwsFileParamsP params
                           );
#endif /* EW_CONFIG_OPTION_FILE_xxx */


/*
 * The following functions extract the corresponding fields from the HTTP/1.0
 * request header, if present, into an application-provided buffer.  Each of
 * these functions return the number of bytes in the actual header value, even
 * if this is larger than the application-provided buffer (though EmWeb/Server
 * will not overwrite the applicatoin buffer by honoring the buffer length).
 * If the header is not present, zero is returned.  The application may
 * sp cify a zero-length buffer in order to determine the size of the header
 * value.  The r turned value begins with the first non-whitespace following
 * the HTTP header and ends just before the terminating end-of-line
```

```
 * character(s).
 */
#ifdef EW_CONFIG_OPTION_CONTEXT_DATE
uintf ewsContextDate ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_DATE */
#ifdef EW_CONFIG_OPTION_CONTEXT_PRAGMA
uintf ewsContextPragma ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_PRAGMA */
#ifdef EW_CONFIG_OPTION_CONTEXT_FROM
uintf ewsContextFrom ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_FROM */
#ifdef EW_CONFIG_OPTION_CONTEXT_IF_MODIFIED_SINCE
uintf ewsContextIfModifiedSince
  ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_IF_MODIFIED_SINCE */
#ifdef EW_CONFIG_OPTION_CONTEXT_REFERER
uintf ewsContextReferer ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_REFERER */
#ifdef EW_CONFIG_OPTION_CONTEXT_USER_AGENT
uintf ewsContextUserAgent ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_USER_AGENT */
#ifdef EW_CONFIG_OPTION_CONTEXT_HOST
uintf ewsContextHost ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CONTEXT_HOST */

#endif /* _EWS_CONTEXT_H */
```

A-2.8. src/include/ews_cgi.h

This file contains the definitions and prototypes of the raw CGI functions of the application interface.

```
/*
 * ews_cgi.h,v 1.22 1997/03/29 17:50:49 ian Exp
 *
 * Product: EmWeb
 * Release: R2_1
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION OF AGRANAT SYSTEMS, INC.
 * THE EMWEB SOFTWARE ARCHITECTURE IS PATENT PENDING.
 * EMWEB IS A TRADEMARK OF AGRANAT SYSTEMS, INC.
 *
 * Copyright (C) 1996, 1997 Agranat Systems, Inc.
 * All Rights Reserved
 *
 * Agranat Systems, Inc.
 * 1345 Main Street
 * Waltham, MA  02154
 *
 * (617) 893-7868
 * sal s@agranat.com, support@agranat.com
 *
```

```
 * http://www.agranat.com/
 *
 * EmWeb/Server application interface to raw CGI
 *
 */
#ifndef _EWS_CGI_H
#define _EWS_CGI_H

#include "ew_types.h"

/*****************************************************************************
 * CGI INTERFACE
 *
 * EmWeb supports a raw CGI interface for imagemap and various special
 * application-defined CGI capabilities.
 *
 * The EmWeb/Compiler creates CGI URLs from specifications provided in
 * _ACCESS files.  The application provides two functions for each CGI URL.
 * First, there is the CGI start function.  This is invoked by the
 * EmWeb/Server to mark the start of a new CGI request.  Second, there is the
 * CGI data function.  This is invoked by the EmWeb/Server one or more times
 * to pass network buffers containing raw CGI request data to the application.
 * The end of request data is indicated by an EWA_NET_BUFFER_NULL EwaNetBuffer.
 *
 * The CGI application uses the ewsCGIData() function to transmit response data
 * to the network.
 *
 *****************************************************************************/

/*
 * ewaCGIStart
 *
 * This per-CGI function is provided by the application.  It is invoked by
 * EmWeb/Server at the start of a new request to the corresponding CGI URL.
 * The application may return an application-specific handle.
 *
 * context      - context of current request
 *
 * Returns application-specific handle to be used by EmWeb/Server in
 * subsequent calls to ewaCGIData.
 */
typedef EwaCGIHandle EwaCGIStart_f ( EwsContext context );

/*
 * ewaCGIData
 *
 * This per-CGI function is provided by the application.  It is invoked by
 * EmWeb/Server zero or more times to pass raw CGI request data to the
 * application.
 *
 * cgi_handle   - application-specific handle as defined by ewaCGIStart
 * buffer       - application buffer containing all or part of the raw CGI
 *                request data
 *
```

```
 * No return value
 */
typedef void EwaCGIData_f ( EwaCGIHandle cgi_handle, EwaNetBuffer buffer );

#ifdef EW_CONFIG_OPTION_CGI

/*
 * ewsCGISendStatus
 * This optional call causes the EmWeb/Server to construct and send an
 * appropriate  HTTP status header (and possibly other server-generated
 * headers).  It must be followed by one or more calls to ewsCGIData() to send
 * additional header information and optional body content.  (Note that a call
 * to ewsCGIData() must not precede the invocation of this function).
 *
 * Note: If the application desires complete control of the response, it may
 * simply send all header information itself using ewsCGIData() instead of
 * invoking this function.
 *
 * context      - context of request
 * status       - 3-digit status code followed by reason string
 * string       - string containing additional headers and/or data, or NULL
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD).
 */
extern EwsStatus ewsCGISendStatus
  ( EwsContext context, const char * status, const char * string );

/*
 * ewsCGIData
 *
 * This function is called by a raw CGI application one or more times to send
 * raw CGI response data to the browser client.  The application is responsible
 * for throttling its use of the CPU.  Note that response data may include raw
 * HTTP headers as well as data.  Therefore, a CR-NL sequence must be
 * transmitted to properly delimit the header portion of the response from
 * any returned data.
 *
 * Note that ewsCGISendStatus() may be called once before any calls to
 * ewsCGIData() to generate standard HTTP headers.  Otherwise, the application
 * must generate all standard headers itself.
 *
 * Note that sending a EWA_NET_BUFFER_NULL terminates the request.
 *
 * context      - context of request
 * buffer       - buffer containing data to send, or EWA_NET_BUFFER_NULL
 *                if this is the last buffer in the response.
 *
 * No return value
 */
extern EwsStatus ewsCGIData ( EwsContext context, EwaNetBuffer buffer );

/*
 *  wsCGIRedirect
 * This call caus s the EmWeb/Server to respond to the browser with the
```

```
 * document specified by the URL (if local), or with a redirect (if not).
 * This function will terminate the request.  Note that this feature can not
 * be used in conjunction with ewsCGIData() or ewsCGIStatus().
 *
 * context      - context of request
 * url          - URL for redirect
 *
 * Returns EWS_STATUS_OK on success, else error code (TBD).
 */
extern EwsStatus ewsCGIRedirect ( EwsContext context, const char *url );

#ifdef EW_CONFIG_OPTION_CGI_SERVER_SOFTWARE
/*
 * ewsCGIServerSoftware
 *
 * Returns a string corresponding to the server software such as
 * "Agranat-EmWeb/R2_1"
 */
extern const char *ewsCGIServerSoftware ( void );
#endif /* EW_CONFIG_OPTION_CGI_SERVER_SOFTWARE */

#ifdef EW_CONFIG_OPTION_CGI_GATEWAY_INTERFACE
/*
 * ewsCGIGatewayInterface
 *
 * Returns a string corresponding to the CGI version "CGI/1.0"
 */
extern const char *ewsCGIGatewayInterface ( void );
#endif /* EW_CONFIG_OPTION_CGI_GATEWAY_INTERFACE */

#ifdef EW_CONFIG_OPTION_CGI_SERVER_PROTOCOL
/*
 * ewsCGIServerProtocol
 *
 * context      - context of request
 * datap        - pointer to data buffer
 * length       - size of data buffer (may be zero)
 *
 * Returns number of bytes in actual protocol string, or zero if not present.
 */
extern uintf ewsCGIServerProtocol
  ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CGI_SERVER_PROTOCOL */

#ifdef EW_CONFIG_OPTION_CGI_REQUEST_METHOD
/*
 * ewsCGIRequestMethod
 *
 * context      - context of request
 * datap        - pointer to data buffer
 * length       - size of data buffer (may be zero)
 *
 * Returns number of bytes in actual request method, or zero if not present.
 */
```

```
extern uintf ewsCGIRequestMethod
  ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CGI_REQUEST_METHOD */

#ifdef EW_CONFIG_OPTION_CGI_PATH_INFO
/*
 * ewsCGIPathInfo
 *
 * context      - context of request
 *
 * Returns pointer to PathInfo string, or NULL if not present
 */
extern const char * ewsCGIPathInfo ( EwsContext context );
#endif /* EW_CONFIG_OPTION_CGI_PATH_INFO */

#ifdef EW_CONFIG_OPTION_CGI_SCRIPT_NAME
/*
 * ewsCGIScriptName
 *
 * context      - context of request
 *
 * Returns pointer to script name string
 */
extern const char * ewsCGIScriptName ( EwsContext context );
#endif /* EW_CONFIG_OPTION_CGI_SCRIPT_NAME */

#ifdef EW_CONFIG_OPTION_CGI_QUERY_STRING
/*
 * ewsCGIQueryString
 *
 * context      - context of request
 * datap        - pointer to data buffer
 * length       - size of data buffer (may be zero)
 *
 * Returns number of bytes in actual query string, or zero if not present.
 */
extern uintf ewsCGIQueryString
  ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CGI_QUERY_STRING */

#ifdef EW_CONFIG_OPTION_CGI_CONTENT_TYPE
/*
 * ewsCGIContentType
 *
 * context      - context of request
 * datap        - pointer to data buffer
 * length       - size of data buffer (may be zero)
 *
 * Returns number of bytes in actual content type, or zero if not present.
 */
ext rn uintf ewsCGICont ntType
  ( EwsContext context, char *datap, uintf l ngth );
#endif /* EW_CONFIG_OPTION_CGI_CONTENT_TYPE */
```

```
#ifdef EW_CONFIG_OPTION_CGI_CONTENT_LENGTH
/*
 * ewsCGIContentLength
 *
 * context      - context of request
 *
 * Returns content length as specified by client, or zero if not present,
 * or EWS_CGI_CONTENT_LENGTH_CHUNKED if chunked.
 */
#define EWS_CGI_CONTENT_LENGTH_CHUNKED  0xffffffff
extern uint32 ewsCGIContentLength ( EwsContext context );
#endif /* EW_CONFIG_OPTION_CGI_CONTENT_LENGTH */

#ifdef EW_CONFIG_OPTION_CGI_CONTENT_ENCODING
/*
 * ewsCGIContentEncoding
 *
 * context      - context of request
 * datap        - pointer to data buffer
 * length       - size of data buffer (may be zero)
 *
 * Returns number of bytes in actual content encoding, or zero if not present.
 */
extern uintf ewsCGIContentEncoding
  ( EwsContext context, char *datap, uintf length );
#endif /* EW_CONFIG_OPTION_CGI_CONTENT_ENCODING */

#endif /* EW_CONFIG_OPTION_CGI */

#endif /* _EWS_CGI_H */
```

TABLE B

```
ew_db.h        Sat Jun 28 07:59:50 1997        1
/*
 * $Id: ew_db.h,v 1.30 1997/06/28 11:59:50 ian Exp $
 *
 * Product: EmWeb
 * Release: $Name:  $
 *
 * %CopyRight%
 *
 * EmWeb database definitions
 *
 */
#ifndef _EW_DB_H
#define _EW_DB_H

#include "ew_common.h"
#include "ew_types.h"
#include "ew_form.h"
#include "ew_imap.h"
#include "ews_def.h"
#include "ews_cgi.h"

/*****************************************************************************
 *
 * DOCUMENT ARCHIVE
 *
 * An EmWeb document archive contains two parts.  These are the archive data
 * component and the archive object component.
 *
 * The archive object component contains all the run-time object code
 * associated with an archive.  A public symbol (determined by EmWeb/Compiler
 * on a per-archive basis) points to an archive descriptor (EwsArchive).  This
 * contains all of the linkage required by the EmWeb/Server to access the
 * archive-specific run-time object code.
 *
 * The archive data component contains all the static data describing
 * individual documents in the archive.  The data component is an endian-
 * independent fully relocatable contiguous block of constant data
 * analogous to a file.  In fact, it could be a file.  The data component
 * begins with an archive header and contains one or more documents.
 *****************************************************************************/

/*
 * Archive Descriptor
 *
 * Each archive has a unique archive descriptor in the object component of the
 * archive.  The archive descriptor is a list of functions called by the
 * EmWeb/Server to execute archive-specific object code while processing
 * requests for documents in the archive.  Unsupported functions (i.e.
 * features not selected when the EmWeb/Compiler was built)  are represented
 * by a NULL pointer.
 */

typedef const void * EmWebString_f     ( EwsContext context, uint32 index );
typedef const char * EmWebInclude_f    ( EwsContext context, uint32 index );

/* Cookie Attribute structure, element of emweb_cookie_tbl[]            */
typedef struct EwCookieAttr_s {
    char *name;
    char *path;
} EwCookieAttributes;


typedef struct EmWebCGITable_s
  {
    EwsCGIStart_f      *start_f;
```

ew_db.h Sat Jun 28 07:59:50 1997 2

```
    EwaCGIData_f      *data_f;
  } EmWebCGITable, *EmWebCGITableP;

typedef struct EwsArchive_s
  {
    uint32      magic;          /* Magic number for archive validation */

#   define EW_ARCHIVE_OBJECT_MAGIC      (('E'<<24)|('W'<<16)|('o'<<8)|'b')

    uint8       version_maj;    /* Archive version (major/minor) */
    uint8       version_min;

#   define EW_ARCHIVE_VERSION_MAJ       1
#   define EW_ARCHIVE_VERSION_MIN       1

    uint8       compiler_maj;   /* EmWeb/Compiler version (major/minor) */
    uint8       compiler_min;

#   define EW_ARCHIVE_DATE_1123_SIZE    32

    char        date_1123[EW_ARCHIVE_DATE_1123_SIZE];   /* creation date */

    /*
     * Archive-specific <EMWEB_STRING> dispatch function.  Takes context
     * and index arguments.  Index selects the C-code fragment corresponding
     * to an instance of <EMWEB_STRING>.
     */
    EmWebString_f       *emweb_string;

    /*
     * Archive-specific <EMWEB_INCLUDE> dispatch function.  Takes context
     * and index arguments.  Index selects the C-code fragment corresponding
     * to an instance of <EMWEB_INCLUDE>.
     */
    EmWebInclude_f      *emweb_include;

    /*
     * Archive-specific form table. The Form node index is hierarchical and
     * contains a form number and an element number.  The form table is
     * indexed by form number (0..n-1).  Each form table entry contains a
     * pointer to an array of field elements indexed by one less than the
     * element number.  (An element number of zero starts the form, and
     * an element number greater than the number of elements in the form
     * ends the form).  The table contains all the necessary information to
     * serve and submit forms.  The emweb_form_enum_table is a list of strings
     * corresponding to RADIO and single SELECT values.
     */
    const EwFormEntry *emweb_form_table;
    uint32       emweb_form_table_size;
    const char   **emweb_form_enum_table;
    uint32       emweb_form_enum_table_size;

    /*
     * Archive-specific imagemap table.  The table represents imagemaps
     * compiled by the EmWeb/Compiler into the archive.  This table is
     * indexed by a document node corresponding to the mapfile.  The
     * table entry contains a table of rectangular regions which are scanned
     * for matching x,y coordinates to find a match.
     */
    const EwImageMapTable *emweb_image_table;
    uint32                emw b_image_table_size;

    /*
     * Archive-specific CGI action dispatch tabl .
     */
```

ew_db.h Sat Jun 28 07:59:50 1997 3

```
    const EmWebCGITable *emweb_cgi_table;
    uint32              emweb_cgi_table_size;

    /*
     * Reference count - this is initialized to zero by the EmWeb/Compiler,
     * but is writable (.data section).  This value is incremented for each
     * cloned page It is also incremented for each page currently in use by an
     * uncompleted request.  This prevents the application from attempting to
     * remove an archive that is in use.
     */
    uint32              reference_count;

    /*
     * Document list - this is the anchor of a list of loaded documents and
     * is used exclusively by EmWeb/Server to keep track of per-document nodes
     * created when loading the archive into the open hashed file system.
     * This should be set to EWS_DOCUMENT_NULL by the EmWeb/Compiler.
     */
    EwsDocument         document_list;

    /*
     * Archive-specific cookie table. This table is indexed by data_offset
     * value find in the cookie header node.
     *
     * !!!NOTE: location of this variable is important. For bacward comatibility
     *          of archives, it MUST be the last field in this structure.
     */
    const EwCookieAttributes *emweb_cookie_tbl;


} EwsArchive_t;

/*
 * DOCUMENT ARCHIVE DATA COMPONENT
 *
 * The document archive data component is a relocatable, endian-independent,
 * static data structure containing compressed documents and information about
 * how to access them.
 *
 * All 16-bit and 32-bit quantities are represented as arrays of octets.  The
 * first octet is the most significant (i.e. Big Endian).
 *
 * All pointers are 32-bit (array of four octets) offsets from the beginning of
 * the archive data.
 *
 * All strings are .asciiz ('\0'-terminated ASCII) and are represented by a
 * pointer to the string.
 *
 * Data storage for strings and various other variable-length components can
 * be placed between headers at the discretion of the EmWeb/Compiler.
 *
 * The document archive data component takes the following form:
 *
 *   EwsArchiveHeader
 *   (optional padding)
 *     { EwsDocumentHeader { EwsDocumentNode, ... } (optional padding) }, ...
 *
 * The optional padding can be used to store variable length fields such
 * as strings, compressed documents, etc.
 */

/*
 * EwsArchiveHeader
 *
 * The archive data component begins with an archive header at offset zero
```

```
ew_db.h        Sat Jun 28 07:59:50 1997        4
 * NOTE WELL: KEEP SIZEOF_EWS_ARCHIVE_HEADER UP TO DATE!!!
 */
typedef struct EwsArchiveHeader_s
  {

    uint8       magic[4];       /* Magic number for archive validation */

#   define EW_ARCHIVE_MAGIC_0   'E'
#   define EW_ARCHIVE_MAGIC_1   'W'
#   define EW_ARCHIVE_MAGIC_2   'd'
#   define EW_ARCHIVE_MAGIC_3   'a'
#   define EW_ARCHIVE_MAGIC     (('E'<<24)|('W'<<16)|('d'<<8)|'a')

    uint8       version_maj;    /* Archive version (major/minor) */
    uint8       version_min;

    uint8       compiler_maj;   /* EmWeb/Compiler version (major/minor) */
    uint8       compiler_min;

    uint8       length[4];      /* length of archive, octets */
    uint8       name_offset[4]; /* Offset to archive name string */
    uint8       date_1123_offset[4]; /* Offset to HTTP date (RFC1123 format) */
    uint8       date_1036_offset[4]; /* Offset to HTTP date (RFC1036 format) */
    uint8       doc_offset[4];  /* Offset to first document header */
    uint8       dict_offset[4]; /* Offset to compression dictionary */

  } EwsArchiveHeader, * EwsArchiveHeaderP;

#define SIZEOF_EWS_ARCHIVE_HEADER 32  /* NEVER use sizeof() */

/*
 * EwsDocumentHeader
 *
 * For each document in the archive, there is a corresponding document
 * header.  The first document header is referenced by the doc_offset
 * field of the archive header.  Subsequent documents are chained together
 * using the next_offset field in each document header.  The last document
 * header in the archive contains a zero in the next_offset field.
 *
 * The following types of documents are supported:
 *
 *   Link       - Generates a redirection URL.  The redirection URL is an
 *                uncompressed string stored in the data area.  node_count
 *                should be zero.
 *
 *   Mime       - Generic (non-text) mime type.  The data area contains the mime
 *                document.  The mime content is indicated by the string at
 *                mime_offset.  Compression is optional.  node_count should be
 *                zero.
 *
 *   Text       - Generic text type.  The text may contain EmWeb/Compiler
 *                HTML extensions for <EMWEB_STRING>, <EMWEB_INCLUDE>,
 *                form get processing, etc.  Each such extension results in
 *                one or more document nodes being present describing the
 *                actions to take at certain points during the processing of
 *                the data.  The data contains the static parts of the
 *                document and may optionally be compressed.  The mime content
 *                is indicated by the string at mime_offset.
 *
 *   Index      - An index URL.  An absolute path URL is stored as an
 *                uncompressed string in the data area.  node_count should be
 *                zero.
 *
 *   Form       - A form action URL.  There should not be a data section.
 *                There will be a FORM node that indicates the index to
```

ew_db.h Sat Jun 28 07:59:50 1997 5

```
 *                use for the start/set/submit operations in handling a
 *                posted form.
 *
 *    CGI         - A CGI URL.  There will be a CGI node that indicates the
 *                index to use for the start/data operations in handling a
 *                raw CGI request.  An optional data section containing
 *                the uncompressed content of the corresponding URL file
 *                (if present) is made available in the EwsContext.  This
 *                could be used for CGI imagemap data, for example.
 *
 *    Imagemap    - A mapfile URL.  There will be an imagemap node that
 *                indicates the index into a per-archive imagemap table
 *                for processing an imagemap.
 *
 * Note that any document might be marked hidden.  If so, the document is
 * not accessible by a client browser directly.  Instead, the document is
 * used as a building block.  It can be cloned, or EMWEB_INCLUDEd.
 *
 * NOTE WELL: KEEP SIZEOF_EWS_DOCUMENT_HEADER UP TO DATE!!
 */
typedef struct EwsDocumentHeader_s
  {
    uint8       next_offset[4]; /* offset to next document or 0 if last */
    uint8       url_offset[4];  /* offset to URL string */

    uint8       document_type;  /* type of document */

    /* If you add a document type, update ../compiler/readarchive.c */
#   define EW_ARCHIVE_DOC_TYPE_MASK     0x0F    /* type field */
#   define EW_ARCHIVE_DOC_TYPE_LINK     0       /* redirection URL */
#   define EW_ARCHIVE_DOC_TYPE_MIME     1       /* generic mime data type */
    /*     EW_ARCHIVE_DOC_TYPE_TEXT     2        * (obsolete) */
#   define EW_ARCHIVE_DOC_TYPE_FORM     4       /* Form action URL */
#   define EW_ARCHIVE_DOC_TYPE_INDEX    5           /* index URL */
#   define EW_ARCHIVE_DOC_TYPE_CGI      8       /* CGI-BIN interface */
#   define EW_ARCHIVE_DOC_TYPE_IMAGEMAP 9       /* Imagemap interface */
#   define EW_ARCHIVE_DOC_TYPE_FILE     10      /* local filesystem */
#   define EW_ARCHIVE_DOC_FLAG_STATIC   0x20    /* treat as static document */
#   define EW_ARCHIVE_DOC_FLAG_FORM     0x40    /* contains form node */
#   define EW_ARCHIVE_DOC_FLAG_HIDDEN   0x80    /* hidden (internal use) */

    uint8       comp_method;    /* compression method */

#   define EW_ARCHIVE_COMPRESSION_NONE  0
#   define EW_ARCHIVE_COMPRESSION_EMWEB 1
#   define EW_ARCHIVE_COMPRESSION_DEFAULT EW_ARCHIVE_COMPRESSION_EMWEB

    uint8       node_count[2];  /* number of document nodes */
    uint8       node_offset[4]; /* offset to array of document nodes */

    uint8       mime_offset[4]; /* offset to mime content type string */
    uint8       realm_offset[4];/* offset to authentication realm string */

    uint8       orig_length[4]; /* length of original (uncompressed) data */
    uint8       data_length[4]; /* length of compressed data */
    uint8       data_offset[4]; /* offset to compressed data */

    uint8       hdr_node_count; /* number of document header nodes */
    uint8       hdr_flags;      /* h ader specific flags              */

#   define EW_ARCHIVE_DOC_HDR_COOKIE_FLG   0x01    /* set if COOKIE hdr present */
#   define EW_ARCHIVE_DOC_HDR_CACHE_FLG    0x02    /* set if CACHE hdr present */
#   define EW_ARCHIVE_DOC_HDR_VARY_FLG     0x03    /* set if VARY hdr present */

    uint8       reserved[2];
```

```
ew_db.h        Sat Jun 28 07:59:50 1997        6
  } EwsDocumentHeader, * EwsDocumentHeaderP;

#define SIZEOF_EWS_DOCUMENT_HEADER 40    /* never use sizeof */

/*
 * ewsDocumentNode
 *
 * The document header points to an array of zero or more document nodes
 * followed by header nodes.
 * The node_count field of the document header indicates the number of nodes
 * present.  Each node represents a special run-time action to be taken by the
 * EmWeb/Server when processing requests for the document.  When processing
 * reaches the specified point in the uncompressed data, the specified action
 * is taken.
 *
 * Header node contains information needed to create HTTP reply header(s).
 * The hdr_count field of the document header indicates the number of hdr_nodes
 * present.  Each node represents a special run-time action to be taken by the
 * EmWeb/Server when processing requests for the document. Header nodes are processed
 * first as the headers are built into the reply befor data part of the document
 * is processed.
 * Header nodes are sorted in the archive by type. Compiler sorts and includes
 * in the archive only headers of the types for which the flag is set in hdr_flags
 * field in the Document Header.
 *
 * The document header node(s) immediately follow the ewsDocumentNodes for a
 * given document.
 * For a cache control header, the first offset in the node will be used to
 * point to the ETAG for this (static) document. ETAGS are used for
 * cache-control in HTTP 1.1. (Note that an ETAG will not be used for a
 * dynamic document).
 * The type field will indicate what type of header node it is - cache or
 * state cookie, etc.
 * The other fields are reserved.
 * Note that in the future, we may use the last 32 bits in this header
 * to be an instance C pointer for the cache control header.

 * NOTE WELL:  KEEP SIZEOF_EWS_DOCUMENT_NODE UP TO DATE!!!
 */
typedef struct EwsDocumentNode_s
  {
    uint8       data_offset[4]; /* location of node in uncompressed data
                                   stream (relative to start of uncompressed
                                   document data),
                                   for COOKIE header it is an index to CookieTable
                                */

    uint8       type;           /* type of node */

    /* If you add a node type, update ../compiler/readarchive.c */
#   define EW_DOCUMENT_NODE_TYPE_STRING          1       /* <EMWEB_STRING> */
#   define EW_DOCUMENT_NODE_TYPE_INCLUDE         2       /* <EMWEB_INCLUDE> */
#   define EW_DOCUMENT_NODE_TYPE_FORM            3       /* <FORM> operation */
#   define EW_DOCUMENT_NODE_TYPE_CGI             4       /* CGI operation */
#   define EW_DOCUMENT_NODE_TYPE_IMAGEMAP        5       /* Image Map */
#   define EW_DOCUMENT_NODE_TYPE_COOKIE          6       /* <EMWEB_COOKIE> */
#   define EW_DOCUMENT_NODE_TYPE_CACHE           7       /* CACHE */
#   define EW_DOCUMENT_NODE_TYPE_VARY            8       /* content negotiation*/

    uint8       attributes;     /* node attributes */

    /* If you add a node attribute, update ../compiler/readarchive.c */
#   define EW_DOCUMENT_NODE_REPEAT      0x01    /* repeat until NULL for
                                                   EMWEB_STRING and
                                                   EMWEB_INCLUDE nodes */
```

```
ew_db.h        Sat Jun 28 07:59:50 1997        7
#   define EW_DOCUMENT_HDR_SECURE        0x02    /* SECURE flag for cookie */

    uint8       reserved[2];     /* reserved for future use */

    uint8       index[4];        /* index assigned to node.  This may be
                                    hierarchical.  (e.g. the FORM index
                                    contains a form number and element
                                    number.  the STRING index contains
                                    a conversion type (most significant
                                    8 bits) and a string number (least
                                    significant 24 bits).
                                    The COOKIE index contains string number
                                    for cookie's value generating C-code.
                                    This index is applied to the switch contained
                                    in emweb_string() function */

  } EwsDocumentNode, * EwsDocumentNodeP;

#define SIZEOF_EWS_DOCUMENT_NODE  12     /* never use sizeof */
#define NEXT_DOCNODE( x ) ((EwsDocumentNodeP)(((uint8 *)(x)) \
                                          + SIZEOF_EWS_DOCUMENT_NODE))

#endif /* _EW_DB_H */
```

ew_form.h Sat Jun 28 07:59:50 1997 1

```
/*
 * $Id: ew_form.h,v 1.23 1997/06/28 11:59:50 ian Exp $
 *
 * Product: EmWeb
 * Release: $Name:  $
 *
 * %CopyRight%
 *
 * EmWeb Form definitions between server and generated code.  These are
 * also used in typed EMWEB_STRINGs, etc.
 *
 */
#ifndef _EW_FORM_H
#define _EW_FORM_H

#include "ew_types.h"
#include "ew_common.h"
#include "ews_def.h"

/*
 * Application Form Serve Function
 * Fills in form template with default values.  (We comment out the prototype
 * to avoid compiler warnings -- the actual application functions define
 * a specific form structure instead of void*).
 *
 * context - request context
 * form    - pointer to form-specific structure
 *
 * No return value
 */
#ifndef __cplusplus
typedef void EwaFormServe_f ( );
#else /* __cplusplus */
typedef void EwaFormServe_f ( EwsContext context, void * form );
#endif /* __cplusplus */

/*
 * Application Form Submit Function
 * Processes submitted form.  (We comment out the prototype
 * to avoid compiler warnings -- the actual application functions define
 * a specific form structure instead of void*).
 *
 * context - request context
 * form    - pointer to form-specific structure
 *
 * Returns NULL for default response to server (accepted), or redirection URL.
 */
#ifndef __cplusplus
typedef char * EwaFormSubmit_f ( );
#else /* __cplusplus */
typedef char * EwaFormSubmit_f ( EwsContext context, void * form );
#endif /* __cplusplus */

/*
 * Form Field Types
 * Each field element has a typ  associated with it.  Each type has conversion
 * routines to handle conversions between HTML and internal representations.
 * NOTE: if you modify this enum list, you MUST update the FieldMap in
 * file ewc_code.c!!
 */
typedef enum EwFieldType_e
  {
    ewFieldTypeDecimalOverride = 0 /* unsigned decimal integer */
   ,ewFieldTypeRadio              /* radio button - serve setup and submit */
   ,ewFieldTypeRadioField         /* radio fields - used in serving form */
```

```
   ,ewFieldTypeSelect           /* select enum - serve setup and submit */
   ,ewFieldTypeSelectField      /* select enum fields - used in serving form */
   ,ewFieldTypeSelectMulti      /* multiple select */
   ,ewFieldTypeCheckbox         /* checkbox */
   ,ewFieldTypeText             /* text field */
   ,ewFieldTypeImage            /* image field */
   ,ewFieldTypeDecimalUint      /* unsigned integer */
   ,ewFieldTypeDecimalInt       /* signed integer */
   ,ewFieldTypeHexInt           /* hexadecimal integer */
   ,ewFieldTypeHexString        /* hexadecimal string */
   ,ewFieldTypeDottedIP         /* dotted IP address */
   ,ewFieldTypeIEEEMAC          /* IEEE MAC */
   ,ewFieldTypeFDDIMAC          /* FDDI MAC */
   ,ewFieldTypeStdMAC           /* Standard MAC */
   ,ewFieldTypeDecnetIV         /* Decnet IV Address */
   ,ewFieldTypeFile             /* input type file handle */
   ,ewFieldTypeTextOverride     /* serve input file name field */
   ,ewFieldTypeObjectID         /* object identifier */
   ,ewFieldTypeMAX              /* MUST BE LAST */
  } EwFieldType;

/*
 * Form Field Representation
 * The compiler generates the following structure for each element in the
 * form.  In general, a FORM document node is placed in the archive at the
 * offset immediately following a VALUE= in TEXT, TEXTAREA, HIDDEN, or PASSWORD
 * field.  The field component of the form node index refers to a field
 * structure as defined below.  The field type would be ewFieldTypeText, or
 * one of the derivative extended types such as ewFieldToDottedIP.  If a
 * VALUE was specified in the source HTML, the compiler translates it into
 * the appropriate internal representation and creates the default_value of
 * the field.  (For ewFieldTypeText, this is simply a pointer to the null
 * terminated string.  For other fields, this is a pointer to a value whose
 * length is determined by the type in ewFieldTable).  The NAME= part of
 * the field is saved, and the offset into the compiler-generated form for
 * the value and status portions of the field is stored in value_offset and
 * status_offset.
 *
 * Checkboxes and multiple selections are fairly straight forward.
 * The type ewFieldCheckbox or ewFieldSelectMulti is used, and the
 * node appears where the word "CHECKED" or "SELECTED" may appear.  If a
 * VALUE= is present, it is saved in the sel_value field.  Otherwise, these
 * types are identical to the extended text types above.
 *
 * Radio buttons and single-select boxes are more complex.  With a single
 * NAME=, there may be multiple VALUE= where only one applies to the actual
 * value of the form field.  With N radio buttons or single select options,
 * the compiler generates N+1 field structures.  One structure is used to
 * read and write the value in the internal generated form.  This would have
 * type ewFieldTypeRadio or ewFieldTypeSelect, contain the default value,
 * offsets into the form structure, and the field name.  In addition,an array of
 * integers containing all values corresponding to the generated enumerator is
 * saved in enum_list.  The other N field structures would represent each radio
 * button or select option and have the type ewFieldTypeRadioField or
 * ewFieldTypeSelectField.  These are referenced by form nodes in the document
 * archive at the location corresponding to where a "CHECKED" or "SELECTED"
 * keyword would be inserted.  They would not have a name field.  Instead,
 * the enum_value field would be set to the VALUE= enumeration corresponding
 * to the specific field.
 */
typedef struct EwFormField_s
  {
    EwFieldType         field_typ ;     /* type of field */
    const void          *default_value; /* pointer to default value */
    uintf               value_offset;   /* offset to value in form struct */
```

```
ew_form.h       Sat Jun 28 07:59:50 1997      3
    uintf               status_offset;  /* offset to status in form struct */
    const char          *name;          /* name of field from NAME="" */
    const int           *enum_list;     /* list of field values by enum */
    int                 enum_value;     /* SELECT/RADIO value of field */
    const char          *sel_value;     /* SELECT multiple value field */
  } EwFormField, * EwFormFieldP;

/*
 * Form Representation
 * The following structure represents an HTML FORM.  This points to the
 * application-provided serve and submit functions, indicates the size
 * of the compiler-generated form structure, and points to a table of
 * field structures as defined above.
 */
typedef struct EwFormEntry_s
  {
    EwaFormServe_f      *serve_f;       /* application serve function */
    EwaFormSubmit_f     *submit_f;      /* application submit function */
    uintf               form_size;      /* sizeof form structure */
    uintf               field_count;    /* number of fields in form */
    const EwFormField   *field_list;     /* list of fields */
  } EwFormEntry, * EwFormEntryP;

/*
 * Document Node Conversion Macros
 * These macros convert between the 32-bit node index and a form and element
 * index.
 */
#define EW_FORM_ELEMENT_START                   (0)
#define EW_FORM_ELEMENT_END                     (0xffff)
#define EW_FORM_NODE_INDEX_TO_FORM_INDEX(i)     ((i) >> 16)
#define EW_FORM_NODE_INDEX_TO_ELEMENT_INDEX(i)  ((i) & 0xffff)
#define EW_FORM_ELEMENT_TO_NODE_INDEX(f, e)     (((f) << 16) | (e))

/*
 * Form conversion functions.
 * The "To" function converts to HTML ASCII format from the internal
 * representation and writes it directly to the output data stream.
 * The "From" function converts from HTML ASCII contained in a parsed network
 * buffer EwsString and converts it to the internal representation.
 */
typedef void EwFieldTypeTo_f
  ( EwsContext context, const EwFormField *field );
typedef void EwFieldTypeFrom_f
  ( EwsContext context, const EwFormField *field, EwsStringP stringp );
typedef void EwFieldTypeFree_f ( void *p );

#ifdef EW_CONFIG_OPTION_FIELDTYPE_RADIO
extern EwFieldTypeTo_f          ewFieldToRadioField;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_RADIO */

#if defined(EW_CONFIG_OPTION_FIELDTYPE_RADIO)\
 || defined(EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE)
extern EwFieldTypeFrom_f        ewFieldFromRadio;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_RADIO | SELECT_SINGLE */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE
extern EwFieldTypeTo_f          ewFieldToSelectField;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE
extern EwFieldTypeTo_f          ewFieldToSelectMulti;
extern EwFieldTypeFrom_f        ewFieldFromSelectMulti;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE */
```

```
ew_form.h        Sat Jun 28 07:59:50 1997        4
#ifdef EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX
extern EwFieldTypeTo_f          ewFieldToCheckbox;
extern EwFieldTypeFrom_f        ewFieldFromCheckbox;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX
extern EwFieldTypeTo_f          ewFieldToText;
extern EwFieldTypeFrom_f        ewFieldFromText;
extern EwFieldTypeFree_f        ewFieldFreeText;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX */

extern EwFieldTypeTo_f          ewFieldToDecimalUint;

#ifdef EW_CONFIG_OPTION_FIELDTYPE_IMAGE
extern EwFieldTypeFrom_f        ewFieldFromImage;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_IMAGE */

#if defined(EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT)\
 || defined(EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT)\
 || defined(EW_CONFIG_OPTION_FIELDTYPE_IMAGE)
extern EwFieldTypeFrom_f        ewFieldFromDecimal;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT|INT|IMAGE */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT
extern EwFieldTypeTo_f          ewFieldToDecimalInt;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_HEX_INT
extern EwFieldTypeTo_f          ewFieldToHexInt;
extern EwFieldTypeFrom_f        ewFieldFromHexInt;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_HEX_INT */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING
extern EwFieldTypeTo_f          ewFieldToHexString;
extern EwFieldTypeFrom_f        ewFieldFromHexString;
extern EwFieldTypeFree_f        ewFieldFreeHexString;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP
extern EwFieldTypeTo_f          ewFieldToDottedIP;
extern EwFieldTypeFrom_f        ewFieldFromDottedIP;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV
extern EwFieldTypeTo_f          ewFieldToDecnetIV;
extern EwFieldTypeFrom_f        ewFieldFromDecnetIV;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV */

#if defined(EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC)\
 || defined(EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC)\
 || defined(EW_CONFIG_OPTION_FIELDTYPE_STD_MAC)
extern EwFieldTypeFrom_f        ewFieldFromMAC;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_IEEE|FDDI|STD_MAC */

#if defined(EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC)\
 || defined(EW_CONFIG_OPTION_FIELDTYPE_STD_MAC)
extern EwFieldTypeTo_f          ewFieldToIEEEMAC;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_IEEE|STD_MAC */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC
extern EwFieldTypeTo_f          ewFieldToFDDIMAC;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_STD_MAC
extern EwFieldTypeFrom_f        ewFieldFromStdMAC;
```

```
ew_form.h       Sat Jun 28 07:59:50 1997       5
#endif /* EW_CONFIG_OPTION_FIELDTYPE_STD_MAC */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_FILE
extern EwFieldTypeFrom_f        ewFieldFromFile;
extern EwFieldTypeFree_f        ewFieldFreeFile;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_FILE */

#ifdef EW_CONFIG_OPTION_FIELDTYPE_OID
extern EwFieldTypeTo_f          ewFieldToObjectID;
extern EwFieldTypeFrom_f        ewFieldFromObjectID;
extern EwFieldTypeFree_f        ewFieldFreeObjectID;
#endif /* EW_CONFIG_OPTION_FIELDTYPE_OID */

typedef struct EwFormFieldTable_s
  {
    uintf                       field_size;     /* size of field */
    EwFieldTypeTo_f             *to_f;          /* convert to HTML */
    EwFieldTypeFrom_f           *from_f;        /* convert from HTML */
    EwFieldTypeFree_f           *free_f;        /* free dynamic pointer */
  } EwFormFieldTable;

#ifdef EW_DEFINE_FIELD_TYPE_TABLE
const EwFormFieldTable ewFormFieldTable[] =
  {

    { sizeof(uint32),  ewFieldToDecimalUint,    NULL,                   NULL }

#   ifdef EW_CONFIG_OPTION_FIELDTYPE_RADIO
    ,{ sizeof(int),     NULL,                   ewFieldFromRadio,       NULL }
    ,{ 0,               ewFieldToRadioField,    NULL,                   NULL }
#   else /* EW_CONFIG_OPTION_FIELDTYPE_RADIO */
    ,{ 0,               NULL,                   NULL,                   NULL }
    ,{ 0,               NULL,                   NULL,                   NULL }
#   endif /* EW_CONFIG_OPTION_FIELDTYPE_RADIO */

#   ifdef EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE
    ,{ sizeof(int),     NULL,                   ewFieldFromRadio,       NULL }
    ,{ 0,               ewFieldToSelectField,   NULL,                   NULL }
#   else /* EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE */
    ,{ 0,               NULL,                   NULL,                   NULL }
    ,{ 0,               NULL,                   NULL,                   NULL }
#   endif /* EW_CONFIG_OPTION_FIELDTYPE_SELECT_SINGLE */

#   ifdef EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE
    ,{ sizeof(boolean), ewFieldToSelectMulti, ewFieldFromSelectMulti,NULL }
#   else /* EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE */
    ,{ 0,               NULL,                   NULL,                   NULL }
#   endif /* EW_CONFIG_OPTION_FIELDTYPE_SELECT_MULTIPLE */

#   ifdef EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX
    ,{ sizeof(boolean), ewFieldToCheckbox,    ewFieldFromCheckbox,      NULL }
#   else /* EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX */
    ,{ 0,               NULL,                   NULL,                   NULL }
#   endif /* EW_CONFIG_OPTION_FIELDTYPE_CHECKBOX */

#   ifdef EW_CONFIG_OPTION_FIELDTYPE_TEXT
    ,{ sizeof(char *),  ewFieldToText,          ewFieldFromText, wFieldFreeText }
#   else /* EW_CONFIG_OPTION_FIELDTYPE_TEXT */
    ,{ 0,               NULL,                   NULL,                   NULL }
#   endif /* EW_CONFIG_OPTION_FIELDTYPE_TEXT */

#   ifdef EW_CONFIG_OPTION_FIELDTYPE_IMAGE
    ,{ sizeof(uint32),  NULL,           ewFieldFromDecimal,  NULL }
#   else /* EW_CONFIG_OPTION_FIELDTYPE_IMAGE */
    ,{ 0,               NULL,                   NULL,                   NULL }
```

```
ew_form.h        Sat Jun 28 07:59:50 1997        6
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_IMAGE */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT
   ,{ sizeof(uint32),    ewFieldToDecimalUint,    ewFieldFromDecimal,   NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_UINT */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT
   ,{ sizeof(uint32),    ewFieldToDecimalInt,     ewFieldFromDecimal,   NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_DECIMAL_INT */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_HEX_INT
   ,{ sizeof(uint32),    ewFieldToHexInt,         ewFieldFromHexInt,    NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_HEX_INT */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_HEX_INT */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING
   ,{ sizeof(EwsFormFieldHexString), ewFieldToHexString, ewFieldFromHexString,
      ewFieldFreeHexString }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_HEX_STRING */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP
   ,{ sizeof(uint32),    ewFieldToDottedIP,       ewFieldFromDottedIP, NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_DOTTEDIP */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC
   ,{ 6,                 ewFieldToIEEEMAC,        ewFieldFromMAC,       NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_IEEE_MAC */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC
   ,{ 6,                 ewFieldToFDDIMAC,        ewFieldFromMAC,       NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_FDDI_MAC */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_STD_MAC
   ,{ 6,                 ewFieldToIEEEMAC,        ewFieldFromStdMAC,    NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_STD_MAC */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_STD_MAC */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV
   ,{ sizeof(uint16),    ewFieldToDecnetIV,       ewFieldFromDecnetIV, NULL }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV */
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_DECNET_IV */

#  ifdef EW_CONFIG_OPTION_FIELDTYPE_FILE
   ,{ sizeof(EwaFileHandle), NULL,          ewFieldFromFile,ewFieldFreeFile }
   ,{ sizeof(char *),    ewFieldToText,     ewFieldFromText,ewFieldFreeText }
#  else  /* !EW_CONFIG_OPTION_FIELDTYPE_FILE */
   ,{ 0,                 NULL,                    NULL,                 NULL }
   ,{ 0,                 NULL,                    NULL,                 NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_FILE */
```

ew_form.h Sat Jun 28 07:59:50 1997 7

```
#  ifdef EW_CONFIG_OPTION_FIELDTYPE_OID
   ,{ sizeof(EwsFormFieldObjectID), ewFieldToObjectID, ewFieldFromObjectID,
      ewFieldFreeObjectID }
#  else /* EW_CONFIG_OPTION_FIELDTYPE_OID */
   ,{ 0,                  NULL,                  NULL,                  NULL }
#  endif /* EW_CONFIG_OPTION_FIELDTYPE_OID */

  };
#else /* EW_DEFINE_FORM_FIELD_TYPE_TABLE */
extern const EwFormFieldTable ewFormFieldTable[];
#endif /* EW_DEFINE_FORM_FIELD_TYPE_TABLE */

#endif /* _EW_FORM_H */
```

```
ew_imap.h       Sat Jun 28 07:58:56 1997    1
/*
 * $Id: ew_imap.h,v 1.7 1997/05/21 21:55:48 ian Exp $
 *
 * Product: EmWeb
 * Release: $Name:  $
 *
 * %CopyRight%
 *
 * EmWeb image map definitions between server and generated code
 *
 */
#ifndef _EW_IMAP_H
#define _EW_IMAP_H

#include "ew_types.h"

typedef struct EwImageMap_s
  {
    uint32      ul_x;   /* upper-left X coordinate */
    uint32      ul_y;   /* upper-left Y coordinate */
    uint32      lr_x;   /* lower-right X coordinate */
    uint32      lr_y;   /* lower-right Y coordinate */
    const char  *url;   /* relative redirection URL */
  } EwImageMap, * EwImageMapP;

typedef struct EwImageMapTable_s
  {
    const EwImageMap *map_table; /* table of map entries */
    uintf       map_entries;     /* size of map_table */
    const char  *default_url;    /* default URL, or NULL if not specified */
  } EwImageMapTable, * EwImageMapTableP;

#endif /* _EW_IMAP_H */
```

CLAIMS

What is claimed is:

1. A method for providing a graphical user interface having dynamic elements, comprising the steps of:

defining elements of the graphical user interface in at least one text document written in a mark-up language;

including at a location in the document a code tag containing a segment of application source code;

serving the text document to a client which interprets the mark-up language; and

when the location is encountered, serving to the client a sequence of characters derived from a result of executing a sequence of instructions represented by the segment of application source code.

2. The method of claim 1, further comprising the step of:

including in the document at least one more code tag containing a segment of application source code.

3. The method of claim 1, wherein the step of defining further comprises the step of:

providing a plurality of documents which collectively define the graphical user interface; and

storing the text document and the plurality of documents as files in a directory tree.

4. The method of claim 3, further comprising the steps of:

compiling the directory tree and the files therein into an archive including content sources; and

decompiling a content source back into the text document before the step of serving.

5. A method for providing a graphical user interface having dynamic elements, comprising the steps of:

defining elements of the graphical user interface in at least one text document written in a mark-up language;

including in the document a string identified by prototype tags;

serving the text document to a prototyping client which interprets the mark-up language but does not recognize and does not display the prototype tags, but does display the string; and

compiling the text document into a content source, omitting the prototype tags and the string identified thereby.

6. The method of claim 5, further comprising the step of:

including at a location in the document a code tag containing a segment of application source code, wherein the prototyping client does not recognize and does not display the code tag; and

the step of further comprising including the segment of application source code in the content source.

7. The method of claim 6, further comprising the steps of:

decompiling the content source into a replica of the text document;

serving the replica to a user client; and

when the location is encountered in the replica, serving to the user client a character stream derived from a result generated by executing the segment of application source code.

8. The method of claim 7, further comprising the steps of:

including in the document at least one more code tag containing a segment of application source code; and

including in the document at least one more string identified by code tags.

9. A method for providing a graphical user interface having dynamic elements, comprising the steps of:

defining elements of the graphical user interface in at least one text document written in a mark-up language;

including at a location in the document a code tag containing a segment of application source code;

including in the document a string identified by prototype tags;

compiling the text document into a content source;

decompiling the content source into a replica of the text document;

serving the replica of the text document to a client which interprets the mark-up language; and

when the location is encountered in the replica, serving to the client a character stream generated by executing the segment of application source code.

10. The method of claim 9, further comprising the steps of:

including in the document at least one more code tag containing a segment of application source code; and

including in the document at least one more string identified by code tags.

11. A software product recorded on a medium, the software product comprising a mark-up language compiler which can compile a mark-up language document into a data structure in a native application programming language, the compiler recognizing one or more code tags which designate included text as a segment of application source code to be saved in a file for compilation by a compiler of the native application programming language.

12. A method for providing a graphical user interface having displayed forms for entry of data, comprising the steps of:

defining elements of the graphical user interface in at least one text document written in a mark-up language;

naming in the document a data item requested of a user and used by an application written in a native application programming language; and

compiling the text document into a content source including a data structure definition in the native application programming language for the named data item.

13. The method of claim 12, wherein the text document is viewed using a prototype client during development and is viewed using a user client during use, further comprising the steps of:

naming the data item within a tag understood in the mark-up language for viewing by both the prototype client and the user client; and

including in the content source a definition of a function to provide a default value for the data item when the content source is served and a definition of a function to perform an application specific operation when the data item is returned by the user client.

14. In a computer-based apparatus for developing a graphical user interface for an application, the apparatus including an editor which can manipulate a document written in a mark-up language and a viewer which can display a document written in the mark-up language, the apparatus further comprising:

a mark-up language compiler which recognizes a code tag containing a source code fragment in a native application source code language, the code tag not otherwise part of the mark-up language, the compiler producing as an output a representation in the native application source code language of the document, including a copy of the source code fragment.

15. The apparatus of claim 14, wherein the mark-up language compiler further recognizes begin and end prototype tags not otherwise part of the mark-up language, the output representation omitting the begin and end prototype tags and all content appearing therebetween in the document.

16. A method for developing and prototyping graphic user interfaces for an application comprising the steps of:

accessing an HTML file,

encapsulating portions of said HTML and entering source code therein,

producing a source module from said HTML with encapsulated portions,

producing source code for a server, and

cross compiling and linking said application, said source code module and said server thereby producing executable object code.

17. The method of claim 16, further comprising the steps of:

running said object code,

executing said compiled encapsulated code when requested by a viewer, wherein said encapsulated code is associated with said application.

18. The method of claim 17, further comprising the steps of:

converting data returned by execution of said compiled encapsulated code into a form displayable by said viewer.

19. The method of claim 18, wherein the data returned by executing said compiled encapsulated code changes over time as a result of changes within the application.

20. A data structure fixed in a computer readable medium, the data structure for use in a computer system including a client and a server in communication with each other, the data structure comprising:

cross-compiled, stored and linked, HTML files with encapsulated portions containing executable code associated with said application, server code, and application code, wherein said executable code is run when the HTML file is served thereby providing real time dynamic data associated with said application.

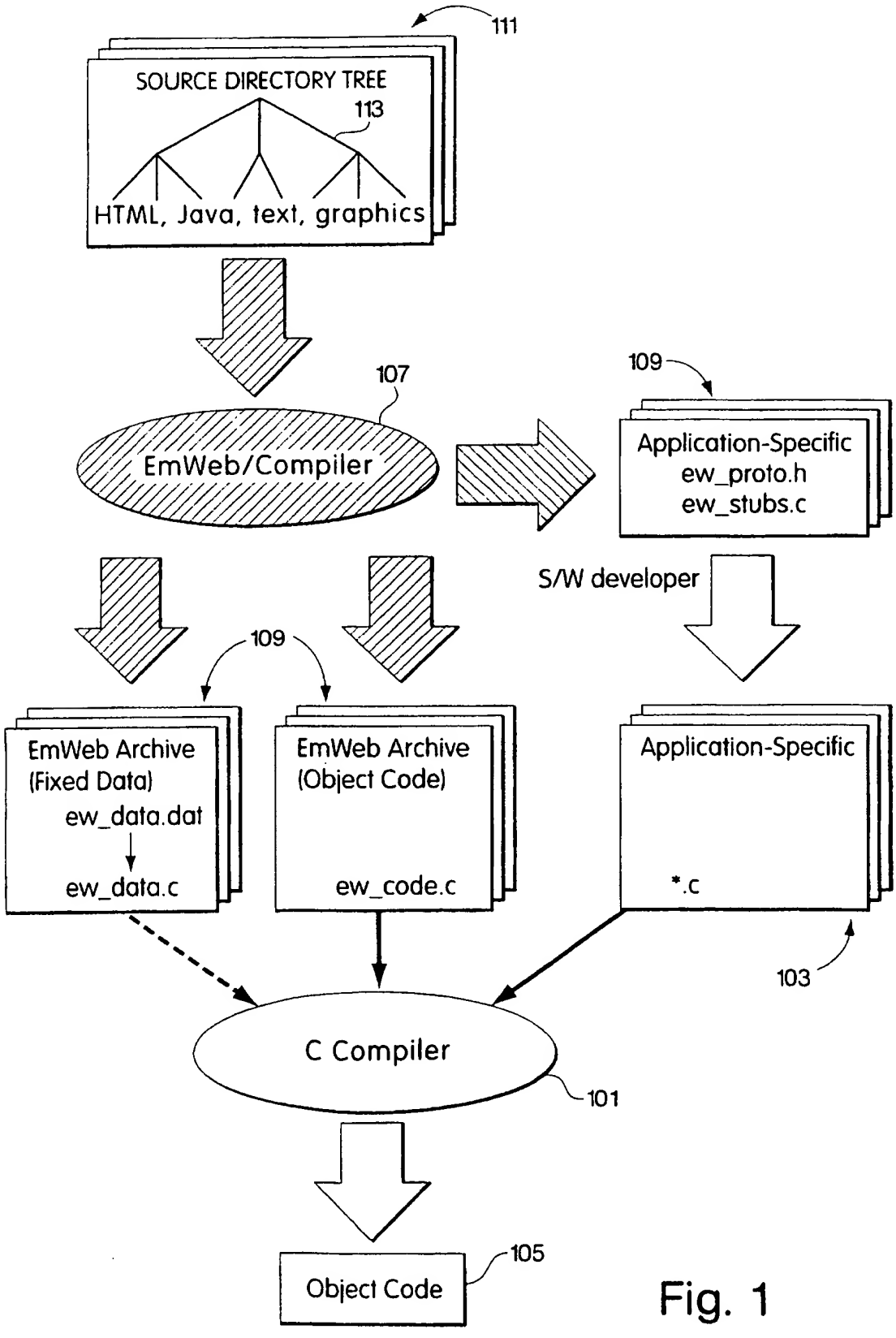

111
SOURCE DIRECTORY TREE
113
HTML, Java, text, graphics
107
EmWeb/Compiler
109
Application-Specific
ew_proto.h
ew_stubs.c
S/W developer
109
EmWeb Archive
(Fixed Data)
ew_data.dat
ew_data.c
EmWeb Archive
(Object Code)
ew_code.c
Application-Specific
*.c
103
C Compiler
101
Object Code
105

Fig. 1

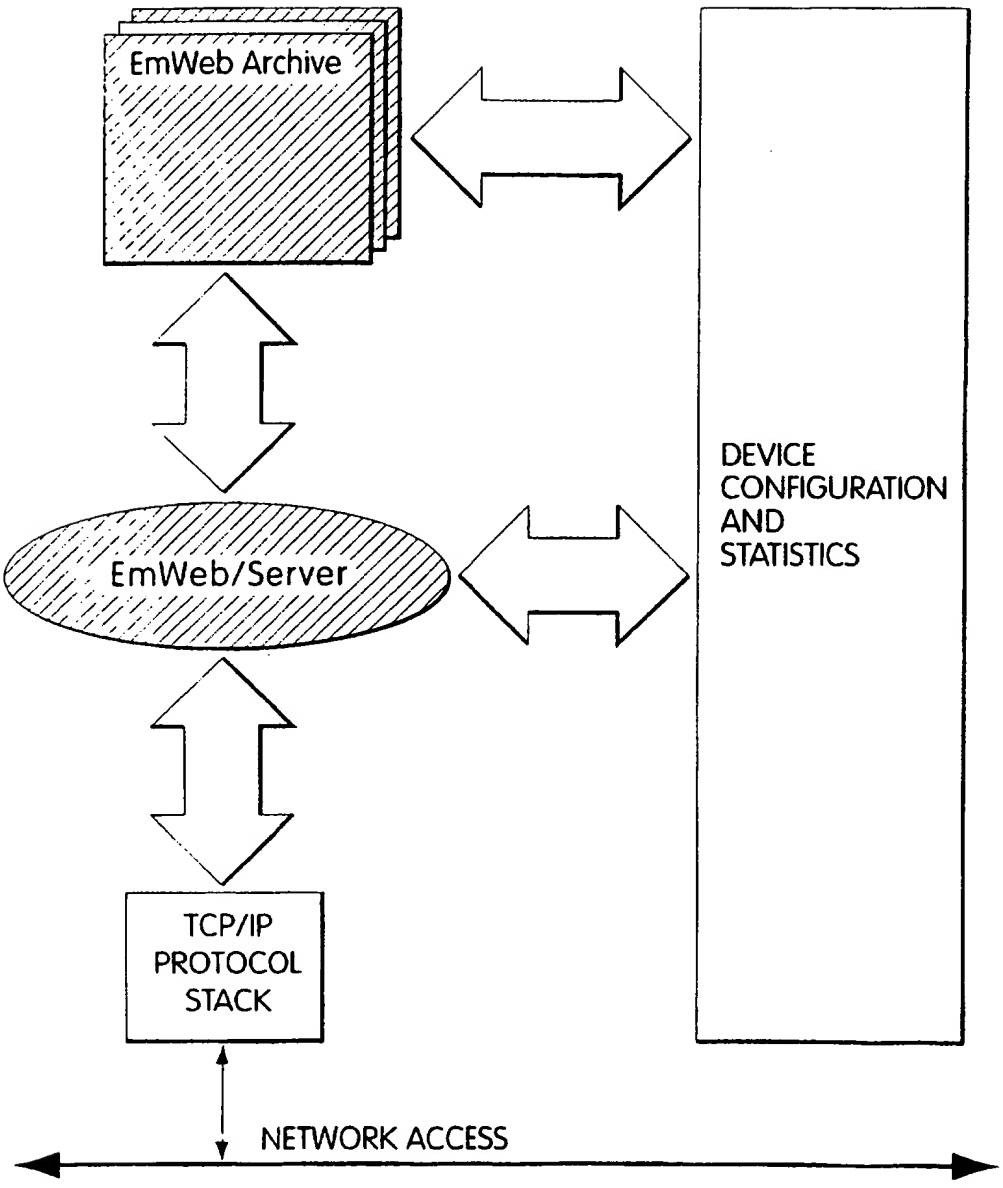

EmWeb Archive
EmWeb/Server
TCP/IP
PROTOCOL
STACK
DEVICE
CONFIGURATION
AND
STATISTICS
NETWORK ACCESS

Fig. 2

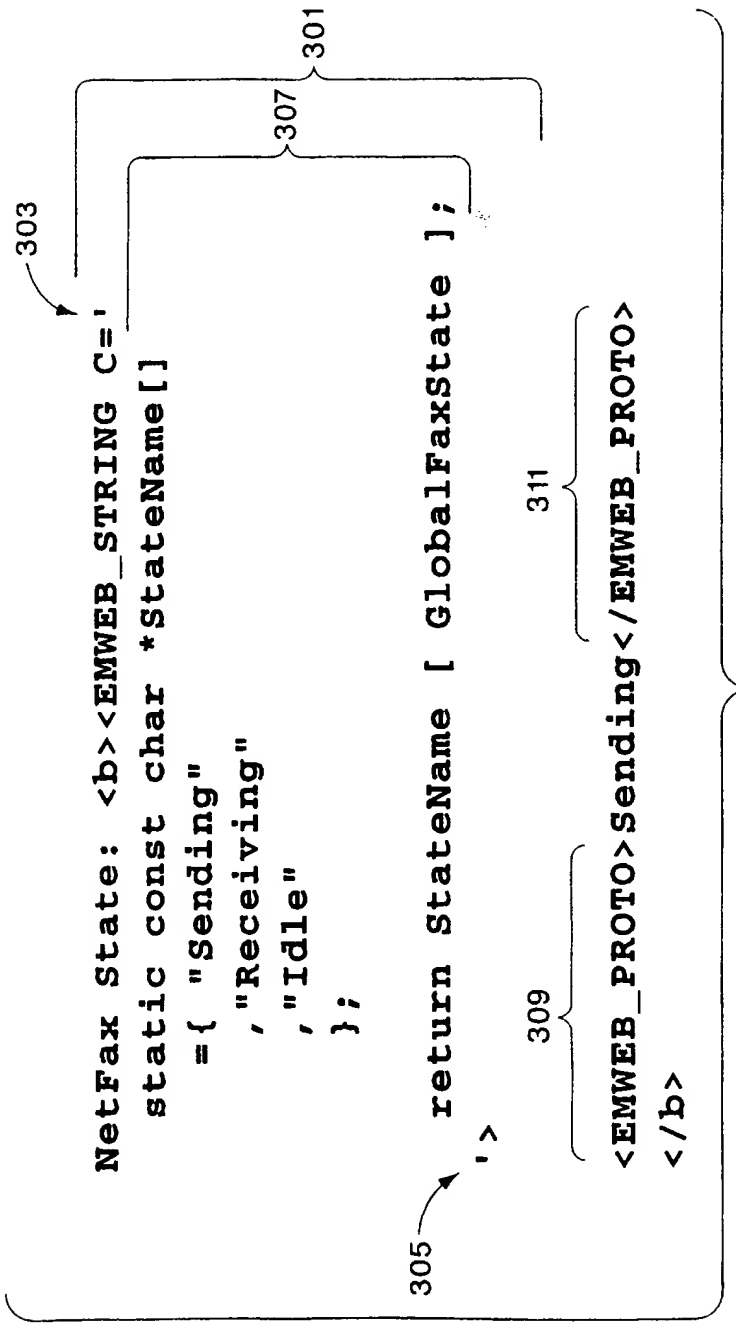

303
301
307
305
309
311
NetFax State: <EMWEB_STRING C='
static const char *StateName[]
={ "Sending"
,"Receiving"
,"Idle"
};
'>
return StateName [GlobalFaxState];
<EMWEB_PROTO>Sending</EMWEB_PROTO>

Fig. 3

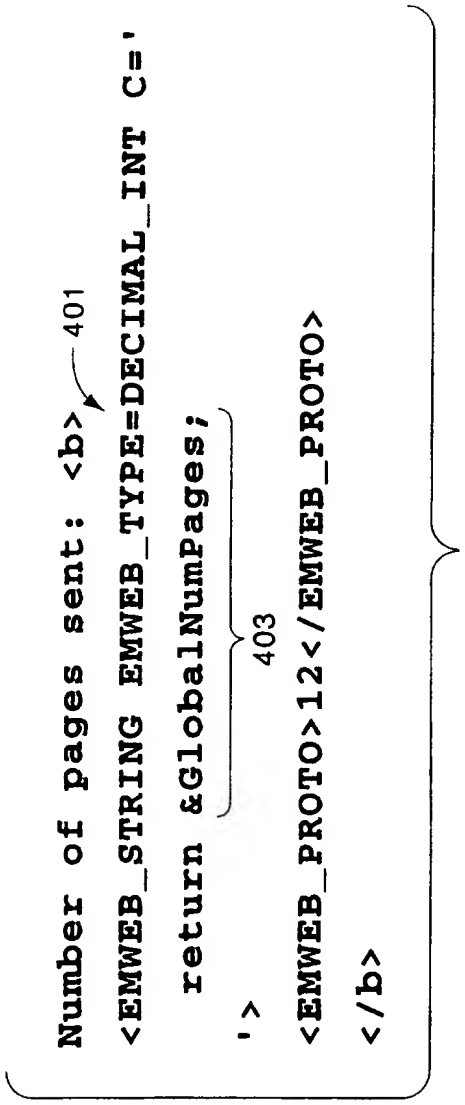

Number of pages sent:
401
<EMWEB_STRING EMWEB_TYPE=DECIMAL_INT C='
return &GlobalNumPages;
'>
403
<EMWEB_PROTO>12</EMWEB_PROTO>

Fig. 4

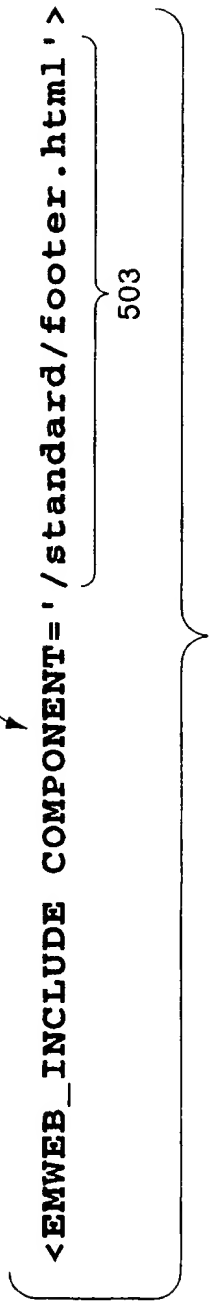


Fig. 5

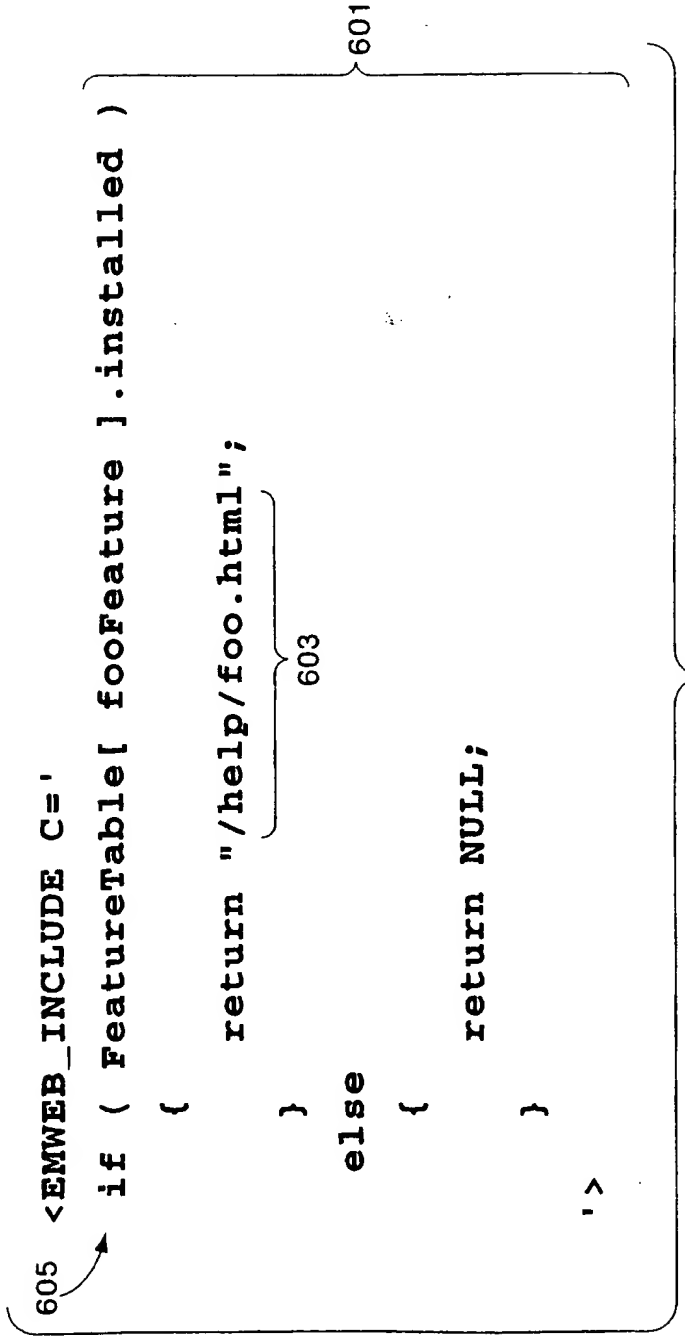
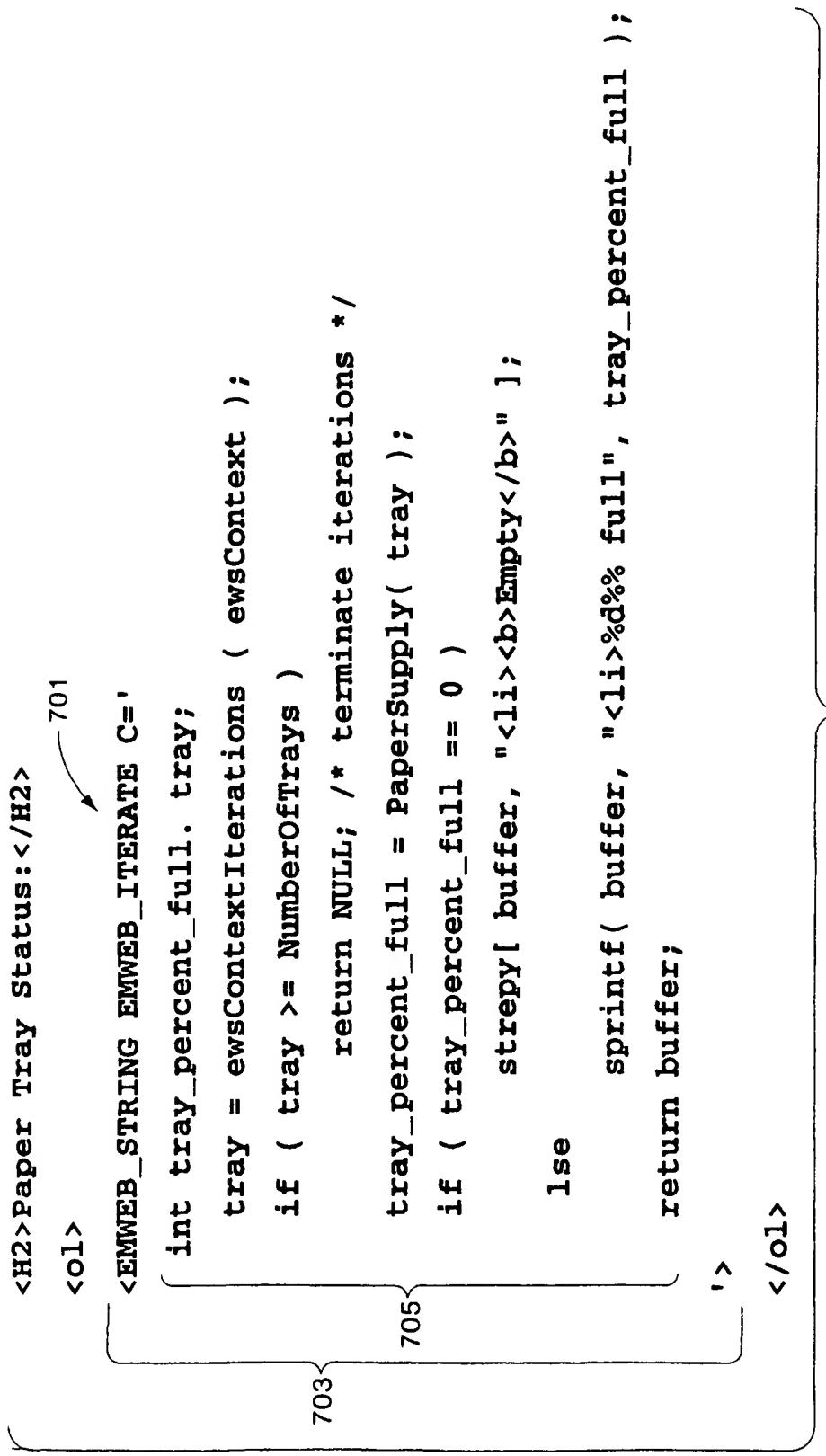

605
<EMWEB_INCLUDE C='
if (FeatureTable[fooFeature].installed)
{
return "/help/foo.html";
}
else
{
return NULL;
}
'>
603
601

Fig. 6


<H2>Paper Tray Status:</H2>

<EMWEB_STRING EMWEB_ITERATE C='
int tray_percent_full. tray;
tray = ewsContextIterations (ewsContext);
if (tray >= NumberOfTrays)
return NULL; /* terminate iterations */
tray_percent_full = PaperSupply(tray);
if (tray_percent_full == 0)
strepy[buffer, "<li>Empty"];
lse
sprintf(buffer, "<li>%d%% full", tray_percent_full);
return buffer;
'>

701
703
705

Fig. 7

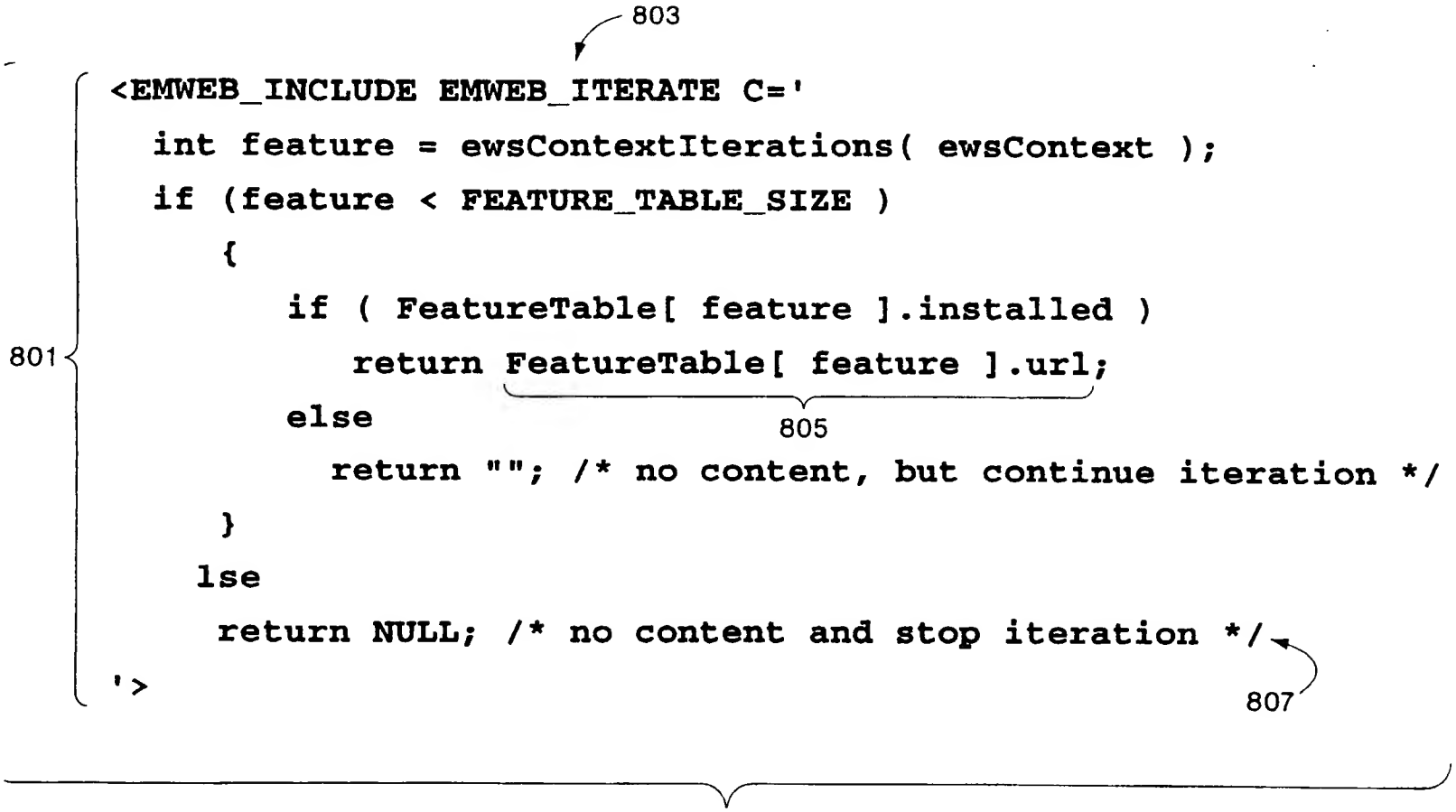

803
<EMWEB_INCLUDE EMWEB_ITERATE C='
int feature = ewsContextIterations(ewsContext);
if (feature < FEATURE_TABLE_SIZE)
{
if (FeatureTable[feature].installed)
return FeatureTable[feature].url;
else
return ""; /* no content, but continue iteration */
}
lse
return NULL; /* no content and stop iteration */
'>
801
805
807

Fig. 8

this HTML:

```
<FORM METHOD=POST EMWEB_NAME=System>
    Fax Server Name:
    <INPUT NAME=sysName TYPE=TEXT>
    <br>
    Fax Log should contain:
    <SELECT NAME =Logging>
       <OPTION VALUE=in_only SELECTED>Incoming Only
       <OPTION VALUE=out_only>Outgoing Only
       <OPTION VALUE=in_out>Incoming and Outgoing
       <OPTION VALUE=no_log>Logging Disabled
</SELECT>
</FORM>
```

911 903 905 901

produces the forms:

Fax Server Name:

Fax Log should contain:

Incoming Only

and generates the structure:

921

```
typedef struct EwaForm_System_s
    {                                          913
    struct                          907
      {                        919       909
        char* sysName;
        EwaFormEnum_ew_archive Logging;
      } value;
    struct                          907
      {
        int8 sysName;
        int8 Logging;
      } status;                          909
    } EwaForm_System, *EwaForm_SystemP;
```

915 917

Fig. 9

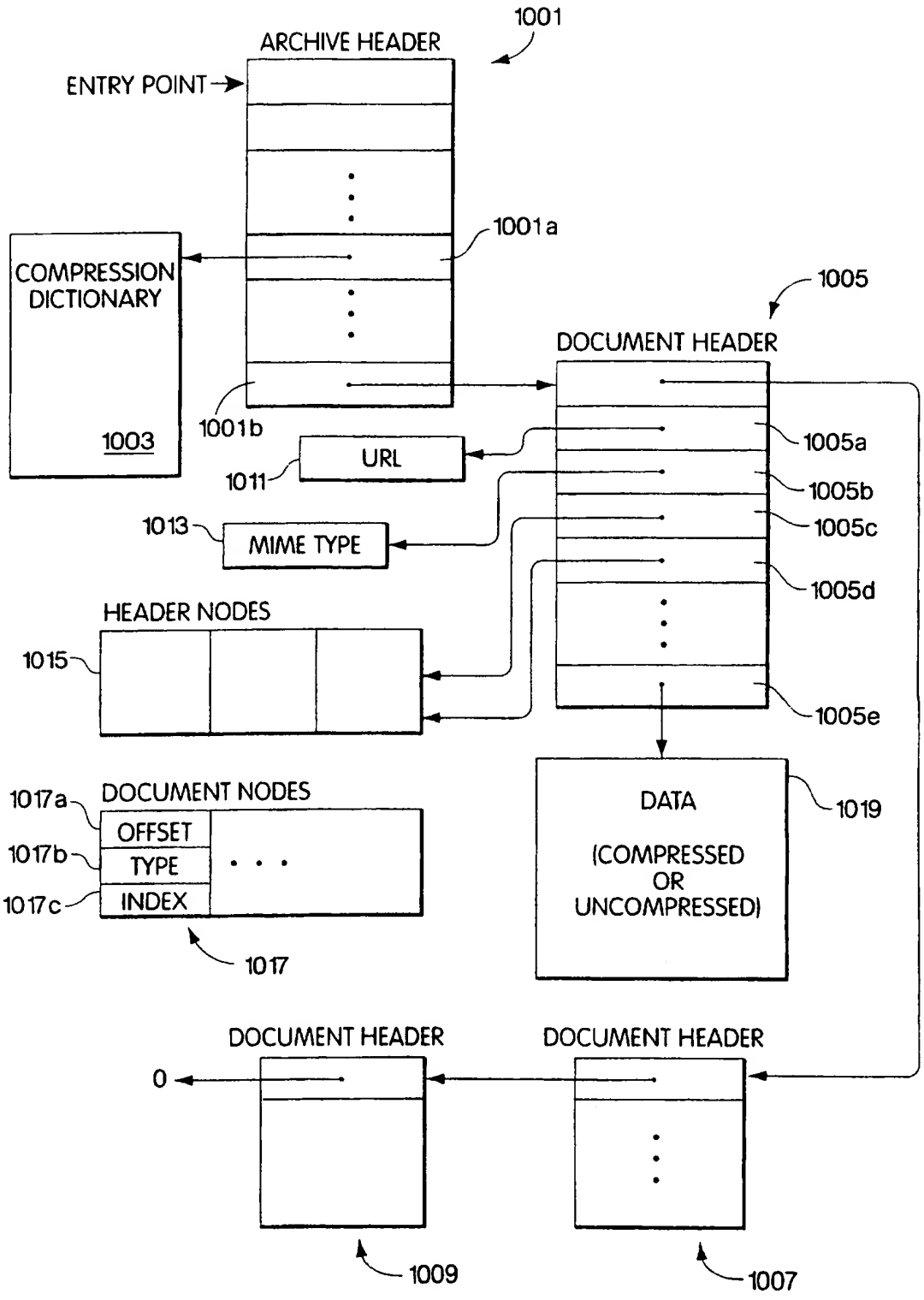

1001
ARCHIVE HEADER
ENTRY POINT
1001a
COMPRESSION DICTIONARY
1003
1001b
1005
DOCUMENT HEADER
1005a
1005b
1005c
1005d
1005e
URL
1011
1013
MIME TYPE
HEADER NODES
1015
DOCUMENT NODES
1017a
OFFSET
1017b
TYPE
1017c
INDEX
1017
DATA
(COMPRESSED OR UNCOMPRESSED)
1019
DOCUMENT HEADER
DOCUMENT HEADER
0
1009
1007

Fig. 10

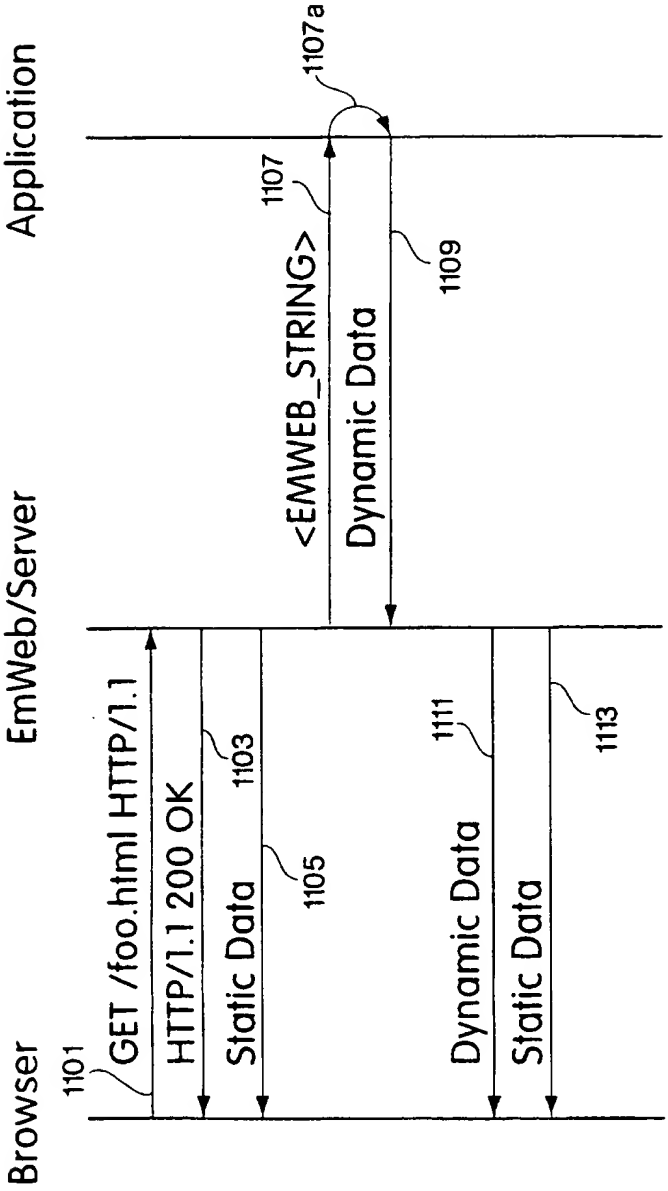

Browser
EmWeb/Server
Application
1101
GET /foo.html HTTP/1.1
HTTP/1.1 200 OK
1103
Static Data
1105
1107
<EMWEB_STRING>
1107a
Dynamic Data
1109
Dynamic Data
1111
Static Data
1113

Fig. 11

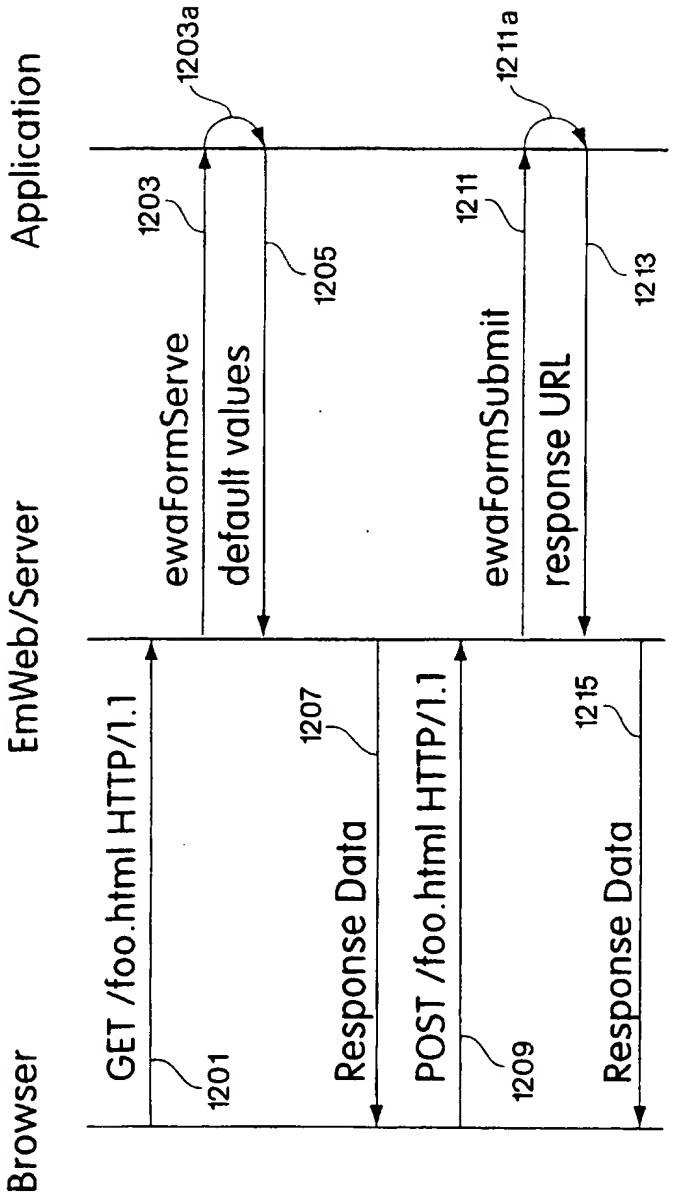

Browser
EmWeb/Server
Application
GET /foo.html HTTP/1.1
1201
ewaFormServe
1203
1203a
default values
1205
Response Data
1207
POST /foo.html HTTP/1.1
1209
ewaFormSubmit
1211
1211a
response URL
1213
Response Data
1215

Fig. 12

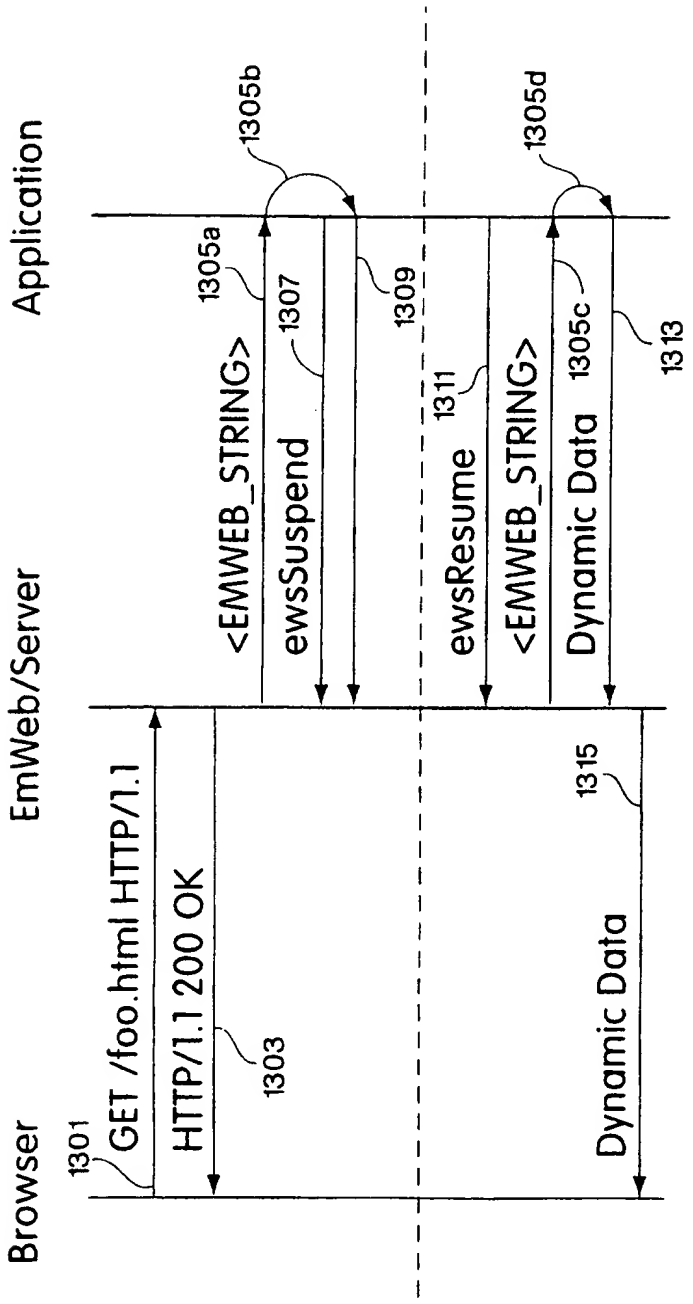

Browser
EmWeb/Server
Application
1301
GET /foo.html HTTP/1.1
HTTP/1.1 200 OK
1303
1305a
<EMWEB_STRING>
1307
ewsSuspend
1309
1305b
1311
ewsResume
1305c
<EMWEB_STRING>
1313
1305d
Dynamic Data
1315
Dynamic Data

Fig. 13